control 4.2.2

Control System Design Tools for GNU Octave
2026-06-07

Table of Contents

1 Overview

1.1 General information

The current GNU Octave control package is based on the work of Lukas F. Reichlin (versions 2 up to 3.0) with major contributions by Thomas Vasileiou. Version 2 of the control package was intended as a replacement of the previous control-1.0.11 by A. Scottedward Hodel and his students. From version 2 onwards the control package is based on the proven open-source library SLICOT.

Its main features are:

- Reliable solvers for Lyapunov, Sylvester and algebraic Riccati equations.
- Pole placement techniques as well as H_2 and H_∞ synthesis methods.
- Frequency-weighted model and controller reduction.
- System identification by subspace methods.
- Overloaded operators due to the use of classes introduced with Octave 3.2.
- Support for descriptor state-space models and non-proper transfer functions.
- Support for multiple systems in time- or frequency-domain plots.
- Improved MATLAB compatibility.

1.2 Using the help function

Some functions of the control package are listed with the somewhat cryptic prefixes `@lti/` or `@iddata/`. These prefixes are only needed to view the help text of the function, e.g. `help norm` shows the built-in function while `help @lti/norm` shows the overloaded function for LTI systems. Note that there are LTI functions like `pole` that have no built-in equivalent. The same is true for IDDATA functions like `nkshift`.

When just using the function, the leading `@lti/` must **not** be typed. Octave selects the right function automatically. So one can type `norm (sys, inf)` and `norm (matrix, inf)` regardless of the class of the argument.

1.3 Distribution

The GNU Octave control package is *free* software. Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. This means that everyone is free to use it and free to redistribute it on certain conditions. The GNU Octave control package is not, however, in the public domain. It is copyrighted and there are restrictions on its distribution, but the restrictions are designed to ensure that others will have the same freedom to use and redistribute Octave that you have. The precise conditions can be found in the GNU General Public License that comes with the GNU Octave control package and that also appears in Appendix A [Copying], page 120.

1.4 Online documentation

Links to online documentation and license information:

- Online documentation of the control package for GNU Octave
- Git repository of the control packag
- Git repository of the used SLICOT-Reference library
- License and copyright information of the used SLICOT-Reference library

2 Function Reference

2.1 Linear Time Invariant Models

2.1.1 @frd/frd

sys = `frd` (*sys*) [Function File]
sys = `frd` (*sys*, *w*) [Function File]
sys = `frd` (*H*, *w*, ...) [Function File]
sys = `frd` (*H*, *w*, *tsam*, ...) [Function File]

Create or convert to frequency response data.

Inputs

sys LTI model to be converted to frequency response data. *sys* can also be a real-valued matrix which is interpreted as continuous-time static gain system. If second argument *w* is omitted, the interesting frequency range is calculated by the zeros and poles of *sys*.

H Frequency response array (p-by-m-by-lw). H(i,j,k) contains the response from input j to output i at frequency k. In the SISO case, a vector (lw-by-1) or (1-by-lw) is accepted as well.

w Frequency vector (lw-by-1) in radian per second [rad/s]. Frequencies must be in ascending order.

tsam Sampling time in seconds. If *tsam* is not specified, a continuous-time model is assumed.

... Optional pairs of properties and values. Type `set (frd)` for more information.

Outputs

sys Frequency response data object.

Option Keys and Values

'H' Frequency response array. See 'Inputs' for details.

'w' Frequency vector. See 'Inputs' for details.

'tsam' Sampling time. See 'Inputs' for details.

'inname' The name of the input channels in *sys*. Cell vector of length m containing strings. Default names are `{'u1', 'u2', ...}`

'outname' The name of the output channels in *sys*. Cell vector of length p containing strings. Default names are `{'y1', 'y2', ...}`

'ingroup' Struct with input group names as field names and vectors of input indices as field values. Default is an empty struct.

'outgroup' Struct with output group names as field names and vectors of output indices as field values. Default is an empty struct.

'name' String containing the name of the model.

'notes' String or cell of string containing comments.

'userdata' Any data type.

See also: dss, @ss/ss, @tf/tf.

2.1.2 @ss/ss

`sys = ss (sys)` [Function File]
`sys = ss (d, ...)` [Function File]
`sys = ss (a, b, ...)` [Function File]
`sys = ss (a, b, c, ...)` [Function File]
`sys = ss (a, b, c, d, ...)` [Function File]
`sys = ss (a, b, c, d, tsam, ...)` [Function File]

Create or convert to state-space model.

Inputs

sys LTI model to be converted to state-space.

a State matrix (n-by-n).

b Input matrix (n-by-m).

c Output matrix (p-by-n). If *c* is empty `[]` or not specified, an identity matrix is assumed.

d Feedthrough matrix (p-by-m). If *d* is empty `[]` or not specified, a zero matrix is assumed.

tsam Sampling time in seconds. If *tsam* is not specified, a continuous-time model is assumed.

... Optional pairs of properties and values. Type `set (ss)` for more information.

Outputs

sys State-space model.

Option Keys and Values

'a', 'b', 'c', 'd', 'e'
State-space matrices. See 'Inputs' for details.

'stname' The name of the states in *sys*. Cell vector containing strings for each state. Default names are `{'x1', 'x2', ...}`

'scaled' Logical. If set to true, no automatic scaling is used, e.g. for frequency response plots.

'tsam' Sampling time. See 'Inputs' for details.

'inname' The name of the input channels in *sys*. Cell vector of length m containing strings. Default names are `{'u1', 'u2', ...}`

'outname' The name of the output channels in *sys*. Cell vector of length p containing strings. Default names are `{'y1', 'y2', ...}`

'ingroup' Struct with input group names as field names and vectors of input indices as field values. Default is an empty struct.

'outgroup' Struct with output group names as field names and vectors of output indices as field values. Default is an empty struct.

'name' String containing the name of the model.

'notes' String or cell of string containing comments.

'userdata' Any data type.

Equations

$$\dot{x} = A\,x + B\,u$$
$$y = C\,x + D\,u$$

Example

```
octave:1> a = [1 2 3; 4 5 6; 7 8 9];
octave:2> b = [10; 11; 12];
octave:3> stname = {'V', 'A', 'kJ'};
octave:4> sys = ss (a, b, 'stname', stname)

sys.a =
         V   A  kJ
   V     1   2   3
   A     4   5   6
   kJ    7   8   9

sys.b =
        u1
   V    10
   A    11
   kJ   12

sys.c =
         V   A  kJ
   y1    1   0   0
   y2    0   1   0
   y3    0   0   1

sys.d =
        u1
   y1    0
   y2    0
   y3    0

Continuous-time model.
octave:5>
```

Compatibility issue

If the state-space model *sys* is converted from a transfer function, the resulting state-space model can be transformed into the form computed by Matlab (a controllable canonical form with flipped state variables order) by using the following similarity transformation:

```
n = size (sys.a, 1)
QSi = inv (ctrb (sys))
T(n,:) = QSi(n,:)
for i=n-1:-1:1, T(i,:) = T(i+1,:)*sys.a, endfor
sys_ml = ss2ss (sys, T)
```

See also: @tf/tf, dss.

2.1.3 @tf/tf

`s = tf ('s')` [Function File]
`z = tf ('z', tsam)` [Function File]

sys = `tf` (*sys*) [Function File]
sys = `tf` (*mat*, ...) [Function File]
sys = `tf` (*num*, *den*, ...) [Function File]
sys = `tf` (*num*, *den*, *tsam*, ...) [Function File]

Create or convert to transfer function model.

Inputs

sys LTI model to be converted to transfer function.

mat Gain matrix to be converted to static transfer function.

num Numerator or cell of numerators. Each numerator must be a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. num{i,j} contains the numerator polynomial from input j to output i. In the SISO case, a single vector is accepted as well.

den Denominator or cell of denominators. Each denominator must be a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. den{i,j} contains the denominator polynomial from input j to output i. In the SISO case, a single vector is accepted as well.

tsam Sampling time in seconds. If *tsam* is not specified, a continuous-time model is assumed.

... Optional pairs of properties and values. Type `set (tf)` for more information.

Outputs

sys Transfer function model.

Option Keys and Values

'num' Numerator. See 'Inputs' for details.

'den' Denominator. See 'Inputs' for details.

'tfvar' String containing the transfer function variable.

'inv' Logical. True for negative powers of the transfer function variable.

'tsam' Sampling time. See 'Inputs' for details.

'inname' The name of the input channels in *sys*. Cell vector of length m containing strings. Default names are `{'u1', 'u2', ...}`

'outname' The name of the output channels in *sys*. Cell vector of length p containing strings. Default names are `{'y1', 'y2', ...}`

'ingroup' Struct with input group names as field names and vectors of input indices as field values. Default is an empty struct.

'outgroup' Struct with output group names as field names and vectors of output indices as field values. Default is an empty struct.

'name' String containing the name of the model.

'notes' String or cell of string containing comments.

'userdata' Any data type.

Example

```
octave:1> s = tf ('s');
octave:2> G = 1/(s+1)

Transfer function 'G' from input 'u1' to output ...

        1
 y1:  -----
      s + 1

Continuous-time model.

octave:3> z = tf ('z', 0.2);
octave:4> H = 0.095/(z-0.9)

Transfer function 'H' from input 'u1' to output ...

       0.095
 y1:  -------
      z - 0.9

Sampling time: 0.2 s
Discrete-time model.

octave:5> num = {[1, 5, 7], [1]; [1, 7], [1, 5, 5]};
octave:6> den = {[1, 5, 6], [1, 2]; [1, 8, 6], [1, 3, 2]};
octave:7> sys = tf (num, den)

Transfer function 'sys' from input 'u1' to output ...

      s^2 + 5 s + 7
 y1:  -------------
      s^2 + 5 s + 6

          s + 7
 y2:  -------------
      s^2 + 8 s + 6

Transfer function 'sys' from input 'u2' to output ...

        1
 y1:  -----
      s + 2

      s^2 + 5 s + 5
 y2:  -------------
      s^2 + 3 s + 2

Continuous-time model.
octave:8>
```

See also: filt, @ss/ss, dss.

2.1.4 dss

sys = dss (*sys*) [Function File]
sys = dss (*d*, ...) [Function File]
sys = dss (*a*, *b*, *c*, *d*, *e*, ...) [Function File]
sys = dss (*a*, *b*, *c*, *d*, *e*, *tsam*, ...) [Function File]

Create or convert to descriptor state-space model.

Inputs

sys LTI model to be converted to state-space.

a State matrix (n-by-n).

b Input matrix (n-by-m).

c Output matrix (p-by-n).

d Feedthrough matrix (p-by-m).

e Descriptor matrix (n-by-n).

tsam Sampling time in seconds. If *tsam* is not specified, a continuous-time model is assumed.

... Optional pairs of properties and values. Type `set (dss)` for more information.

Outputs

sys Descriptor state-space model.

Option Keys and Values

'a', 'b', 'c', 'd', 'e'
State-space matrices. See 'Inputs' for details.

'stname' The name of the states in *sys*. Cell vector containing strings for each state. Default names are `{'x1', 'x2', ...}`

'scaled' Logical. If set to true, no automatic scaling is used, e.g. for frequency response plots.

'tsam' Sampling time. See 'Inputs' for details.

'inname' The name of the input channels in *sys*. Cell vector of length m containing strings. Default names are `{'u1', 'u2', ...}`

'outname' The name of the output channels in *sys*. Cell vector of length p containing strings. Default names are `{'y1', 'y2', ...}`

'ingroup' Struct with input group names as field names and vectors of input indices as field values. Default is an empty struct.

'outgroup' Struct with output group names as field names and vectors of output indices as field values. Default is an empty struct.

'name' String containing the name of the model.

'notes' String or cell of string containing comments.

'userdata' Any data type.

Equations

$$E\,\dot{x} = A\,x + B\,u$$

$$y = C\,x + D\,u$$

See also: @ss/ss, @tf/tf.

2.1.5 filt

`sys = filt (num, den, ...)` [Function File]
`sys = filt (num, den, tsam, ...)` [Function File]

Create discrete-time transfer function model from data in DSP format.

Inputs

num Numerator or cell of numerators. Each numerator must be a row vector containing the coefficients of the polynomial in ascending powers of z^-1. num{i,j} contains the numerator polynomial from input j to output i. In the SISO case, a single vector is accepted as well.

den Denominator or cell of denominators. Each denominator must be a row vector containing the coefficients of the polynomial in ascending powers of z^-1. den{i,j} contains the denominator polynomial from input j to output i. In the SISO case, a single vector is accepted as well.

tsam Sampling time in seconds. If *tsam* is not specified, default value -1 (unspecified) is taken.

... Optional pairs of properties and values. Type `set (filt)` for more information.

Outputs

sys Discrete-time transfer function model.

Option Keys and Values

'num' Numerator. See 'Inputs' for details.

'den' Denominator. See 'Inputs' for details.

'tfvar' String containing the transfer function variable.

'inv' Logical. True for negative powers of the transfer function variable.

'tsam' Sampling time. See 'Inputs' for details.

'inname' The name of the input channels in *sys*. Cell vector of length m containing strings. Default names are `{'u1', 'u2', ...}`

'outname' The name of the output channels in *sys*. Cell vector of length p containing strings. Default names are `{'y1', 'y2', ...}`

'ingroup' Struct with input group names as field names and vectors of input indices as field values. Default is an empty struct.

'outgroup' Struct with output group names as field names and vectors of output indices as field values. Default is an empty struct.

'name' String containing the name of the model.

'notes' String or cell of string containing comments.

'userdata' Any data type.

Example

$$H(z^{-1}) = \frac{3z^{-1}}{1 + 4z^-1 + 2z^-2}$$

```
octave:1> H = filt ([0, 3], [1, 4, 2])   # or filt ([0, 3, 0], [1, 4, 2])

Transfer function 'H' from input 'u1' to output ...

            3 z^-1
 y1:  -------------------
      1 + 4 z^-1 + 2 z^-2

Sampling time: unspecified
Discrete-time model.
```

See also: @tf/tf.

2.1.6 zpk

s = zpk (*'s'*) [Function File]
z = zpk (*'z'*, *tsam*) [Function File]
sys = zpk (*sys*) [Function File]
sys = zpk (*k*, ...) [Function File]
sys = zpk (*z*, *p*, *k*, ...) [Function File]
sys = zpk (*z*, *p*, *k*, *tsam*, ...) [Function File]
sys = zpk (*z*, *p*, *k*, *tsam*, ...) [Function File]

Create transfer function model from zero-pole-gain data. This is just a stop-gap compatibility wrapper since zpk models are not yet implemented.

Inputs

sys LTI model to be converted to transfer function.

z Cell of vectors containing the zeros for each channel. z{i,j} contains the zeros from input j to output i. In the SISO case, a single vector is accepted as well.

p Cell of vectors containing the poles for each channel. p{i,j} contains the poles from input j to output i. In the SISO case, a single vector is accepted as well.

k Matrix containing the gains for each channel. k(i,j) contains the gain from input j to output i.

tsam Sampling time in seconds. If *tsam* is not specified, a continuous-time model is assumed.

... Optional pairs of properties and values. Type `set (tf)` for more information.

Outputs

sys Transfer function model.

See also: @tf/tf, @ss/ss, dss, @frd/frd.

2.2 Model Data Access

2.2.1 @lti/get

get (*sys*) [Function File]
value = get (*sys*, *"key"*) [Function File]
[*val1*, *val2*, ...] = get (*sys*, *"key1"*, *"key2"*, ...) [Function File]

Access key values of LTI objects.

2.2.2 @lti/set

`set (sys)` [Function File]
`set (sys, "key", value, ...)` [Function File]
`retsys = set (sys, "key", value, ...)` [Function File]

Set or modify properties of LTI objects. If no return argument *retsys* is specified, the modified LTI object is stored in input argument *sys*. `set` can handle multiple properties in one call: `set (sys, 'key1', val1, 'key2', val2, 'key3', val3)`. `set (sys)` prints a list of the object's key names.

2.2.3 dssdata

`[a, b, c, d, e, tsam] = dssdata (sys)` [Function File]
`[a, b, c, d, e, tsam] = dssdata (sys, [])` [Function File]

Access descriptor state-space model data.

Argument *sys* is not limited to descriptor state-space models. If *sys* is not a descriptor state-space model, it is converted automatically. *sys* can also be a real-valued gain matrix which is then interpreted as static gain system.

Inputs

sys Any type of LTI model or a real-valued matrix which is interpreted as continous-time static gain.

[] In case *sys* is not a dss model (descriptor matrix *e* empty), `dssdata (sys, [])` returns the empty element `e = []` whereas `dssdata (sys)` returns the identity matrix `e = eye (size (a))`.

Outputs

a State matrix (n-by-n).

b Input matrix (n-by-m).

c Measurement matrix (p-by-n).

d Feedthrough matrix (p-by-m).

e Descriptor matrix (n-by-n).

tsam Sampling time in seconds. If *sys* is a continuous-time model, a zero is returned.

2.2.4 filtdata

`[num, den, tsam] = filtdata (sys)` [Function File]
`[num, den, tsam] = filtdata (sys, "vector")` [Function File]

Access discrete-time transfer function data in DSP format. Argument *sys* is not limited to transfer function models. If *sys* is not a transfer function, it is converted automatically. *sys* can also be a real-valued gain matrix which is then interpreted as static gain system with unspecified sampling time (-1).

Inputs

sys Any type of discrete-time LTI model or a real-valued matrix which is interpreted as discrete-time static gain with unspecified sampling time -1.

"*v*", "*vector*"
For SISO models, return *num* and *den* directly as column vectors instead of cells containing a single column vector.

Outputs

num Cell of numerator(s). Each numerator is a row vector containing the coefficients of the polynomial in ascending powers of z^-1. num{i,j} contains the numerator polynomial from input j to output i. In the SISO case, a single vector is possible as well.

den Cell of denominator(s). Each denominator is a row vector containing the coefficients of the polynomial in ascending powers of z^-1. den{i,j} contains the denominator polynomial from input j to output i. In the SISO case, a single vector is possible as well.

tsam Sampling time in seconds. If *tsam* is not specified, -1 is returned.

2.2.5 frdata

`[H, w, tsam] = frdata (sys)` [Function File]
`[H, w, tsam] = frdata (sys, "vector")` [Function File]

Access frequency response data.

Argument *sys* is not limited to frequency response data objects. If *sys* is not a frd object, it is converted automatically.

Inputs

sys Any type of LTI model or a real-valued matrix which is interpreted as continous-time static gain.

"*v*", "*vector*"
In case *sys* is a SISO model, this option returns the frequency response as a column vector (lw-by-1) instead of an array (p-by-m-by-lw).

Outputs

H Frequency response array (p-by-m-by-lw). H(i,j,k) contains the response from input j to output i at frequency k. In the SISO case, a vector (lw-by-1) is possible as well, see input "*vector*".

w Frequency vector (lw-by-1) in radian per second [rad/s]. Frequencies are in ascending order.

tsam Sampling time in seconds. If *sys* is a continuous-time model, a zero is returned.

2.2.6 ssdata

`[a, b, c, d, tsam] = ssdata (sys)` [Function File]

Access state-space model data. Argument *sys* is not limited to state-space models. If *sys* is not a state-space model, it is converted automatically. *sys* can also be a real-valued matrix which is then interpreted as continuous-time static gain system.

Inputs

sys Any type of LTI model or a real-valued matrix which is interpreted as continous-time static gain.

Outputs

a State matrix (n-by-n).

b Input matrix (n-by-m).

c Measurement matrix (p-by-n).

d Feedthrough matrix (p-by-m).

tsam Sampling time in seconds. If *sys* is a continuous-time model, a zero is returned.

Compatibility issue

If *sys* is given by an input-output description, like, e.g., a transfer function, the resulting state-space model has a different form than the one provided by Matlab, see Section 2.1.2 [@ss/ss], page 3 for details.

2.2.7 tfdata

`[num, den, tsam] = tfdata (sys)` [Function File]
`[num, den, tsam] = tfdata (sys, "vector")` [Function File]
`[num, den, tsam] = tfdata (sys, "tfpoly")` [Function File]

Access transfer function data. Argument *sys* is not limited to transfer function models. If *sys* is not a transfer function, it is converted automatically. *sys* can also be a real-valued gain matrix which is then interpreted as static gain system.

Inputs

sys Any type of LTI model or a real-valued matrix which is interpreted as continous-time static gain.

"*v*", "*vector*"
For SISO models, return *num* and *den* directly as column vectors instead of cells containing a single column vector.

"*t*", "*tfpoly*"
Return the polynomials as tfpoly class.

Outputs

num Cell of numerator(s). Each numerator is a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. num{i,j} contains the numerator polynomial from input j to output i. In the SISO case, a single vector is possible as well.

den Cell of denominator(s). Each denominator is a row vector containing the coefficients of the polynomial in descending powers of the transfer function variable. den{i,j} contains the denominator polynomial from input j to output i. In the SISO case, a single vector is possible as well.

tsam Sampling time in seconds. If *sys* is a continuous-time model, a zero is returned.

2.2.8 zpkdata

`[z, p, k, tsam] = zpkdata (sys)` [Function File]
`[z, p, k, tsam] = zpkdata (sys, "v")` [Function File]

Access zero-pole-gain data. *sys* can also be a real-valued matrix which is then interpreted as continuous-time static gain system.

Inputs

sys Any type of LTI model or a real-valued matrix which is interpreted as continous-time static gain.

"*v*", "*vector*"
For SISO models, return *z* and *p* directly as column vectors instead of cells containing a single column vector.

Outputs

z Cell of column vectors containing the zeros for each channel. z{i,j} contains the zeros from input j to output i.

p Cell of column vectors containing the poles for each channel. p{i,j} contains the poles from input j to output i.

k Matrix containing the gains for each channel. k(i,j) contains the gain from input j to output i.

tsam Sampling time in seconds. If *sys* is a continuous-time model, a zero is returned.

2.3 Model Conversions

2.3.1 @lti/c2d

sys = c2d (*sys*, *tsam*) [Function File]
sys = c2d (*sys*, *tsam*, *method*) [Function File]
sys = c2d (*sys*, *tsam*, *'prewarp'*, *w0*) [Function File]

Convert the continuous LTI model into its discrete-time equivalent.

Inputs

sys Continuous-time LTI model.

tsam Sampling time in seconds.

method Optional conversion method. If not specified, default method "*zoh*" is taken.

- *'impulse'* Impulse Invarient transformation.
- *'zoh'* Zero-order hold or matrix exponential.
- *'foh'* First-order hold, linear approximation of the input signals between two sample times
- *'tustin'*, *'bilin'* Bilinear transformation or Tustin approximation.
- *'prewarp'* Bilinear transformation with pre-warping at frequency *w0*.
- *'matched'* Matched pole/zero method.

Outputs

sys Discrete-time LTI model.

2.3.2 @lti/d2c

sys = d2c (*sys*) [Function File]
sys = d2c (*sys*, *method*) [Function File]
sys = d2c (*sys*, *'prewarp'*, *w0*) [Function File]

Convert the discrete LTI model into its continuous-time equivalent.

Inputs

sys Discrete-time LTI model.

method Optional conversion method. If not specified, default method "*zoh*" is taken.

- *'zoh'* Zero-order hold or matrix logarithm.
- *'tustin'*, *'bilin'* Bilinear transformation or Tustin approximation.
- *'prewarp'* Bilinear transformation with pre-warping at frequency *w0*.
- *'matched'* Matched pole/zero method.

Outputs

sys Continuous-time LTI model.

2.3.3 @lti/d2d

`sys = d2d (sys, tsam)` [Function File]
`sys = d2d (sys, tsam, method)` [Function File]
`sys = d2d (sys, tsam, 'prewarp', w0)` [Function File]

Resample discrete-time LTI model to sampling time *tsam*.

Inputs

sys Discrete-time LTI model.

tsam Desired sampling time in seconds.

method Optional conversion method. If not specified, default method "*zoh*" is taken.

- *'zoh'* Zero-order hold or matrix logarithm.
- *'tustin'*, *'bilin'* Bilinear transformation or Tustin approximation.
- *'prewarp'* Bilinear transformation with pre-warping at frequency *w0*.
- *'matched'* Matched pole/zero method.

Outputs

sys Resampled discrete-time LTI model with sampling time *tsam*.

2.3.4 @lti/prescale

`[scaledsys, info] = prescale (sys)` [Function File]

Scale state-space model. The scaled model *scaledsys* is equivalent to *sys*, but the state vector is scaled by diagonal transformation matrices in order to increase the accuracy of subsequent numerical computations. Frequency response commands perform automatic scaling unless model property *scaled* is set to *true*.

Inputs

sys LTI model.

Outputs

scaledsys Scaled state-space model.

info Structure containing additional information.

info.SL Left scaling factors. `Tl = diag (info.SL)`.

info.SR Right scaling factors. `Tr = diag (info.SR)`.

Equations

$$E_s = T_l \, E \, T_r$$

$$A_s = T_l \, A \, T_r$$

$$B_s = T_l \, B$$

$$C_s = C \, T_r$$

$$D_s = D$$

For proper state-space models, *Tl* and *Tr* are inverse of each other.

Algorithm
Uses SLICOT TB01ID and TG01AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.3.5 @lti/xperm

retsys = xperm (*sys*, *idx*) [Function File]

Reorder states in state-space models.

Inputs

sys State-space model.

idx Vector containing the state indices in the desired order. Alternatively, a cell vector containing the state names is possible as well. See `sys.stname`. State names only work if they were assigned explicitly before, i.e. `sys.stname` contains no empty strings. Note that if certain state indices of *sys* are missing or appear multiple times in *idx*, these states will be pruned or duplicated accordingly in the resulting state-space model *retsys*.

Outputs

retsys Resulting state-space model with states reordered according to *idx*.

2.3.6 @ss/ss2ss

SYS_T = ss2ss (*SYS*, *T*) [Function File]

[*A_T* *B_T* *C_T* *D_T*] = ss2ss (*A*, *B*, *C*, *D*, *T*) [Function File]

Applies the similarity transformation T to a state-space model

Given the state space model

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

and the transformation matrix T, which maps the state vector x to another coordinate system

$$\bar{x} = Tx$$

the state-space model is transformed in a way that results in an equivalent state-space model which is based on the new state vector

$$\dot{\bar{x}} = TAT^{-1}\bar{x} + TBu$$

$$y = CT^{-1}\bar{x} + Du$$

Please note: In the literature, T may be defined inversely:

$$\bar{x} = T^{-1}x$$

References:

Control System Design, page 484 by Goodwin, Graebe, Salgado, 2000

2.3.7 bdschur

[*T*, *S*, *BLKSZ*] = bdschur (*A*) [Function File]

[*T*, *S*, *BLKSZ*] = bdschur (*A*, *condmax*) [Function File]

Compute a block-diagonal Schur decomposition.

Inputs

A Matrix to be decomposed.

condmax Maximum condition number (real scalar value greater or equal 1) of the resulting transformation matrix *T*. This parameter is optional. If not provided, a value of 1e4 is used.

Outputs

T Transformation matrix.

S Block diagonalized matrix.

BLKSZ Array with sizes of the blocks in *S*.

The resulting matrix *S* in block diagonal form is given by

```
        -1
S = T  A T
```

2.3.8 pade

pade (*T*, *n*) [Function File]
pade (*T*, *n*, *m*) [Function File]
pade (*T*, *n1*, *m1*, *n2*, *m2*, ..., *nk*, *mk*) [Function File]
sys = pade (...) [Function File]
sys = pade (...) [Function File]
[*num*, *den*] = pade (...) [Function File]
[*num*, *den*] = pade (...) [Function File]

Calculate Padé approximation of a dead-time by zeros and poles

Inputs

T Dead-time to be approximated.

n Number of poles of the approximations.

m Number of zeros of the approximation, If omitted, the number of zeros is the same as the number *n* of poles.

More then one approximation can be requested by providing the pairs pairs *n1*, *m1*, *n2*, *m2*, ..., *nk*, *mk*.

Outputs

sys LTI system with the poles and zeros of the Padé approximation. *sys* is given as transfer function. If more than one approximation is requested, *sys* is a cell array of LTI systems.

num Numerator polynomial of the resulting transfer function. If more than one approximation is requested, *num* is a cell array of numerator polynomials.

den Denominator polynomial of the resulting transfer function. If more than one approximation is requested, *den* is a cell array of denominator polynomials.

If no output argument is requested, the step response and the bode diagram of the approximatons with orders *n1*, *m1* to *nk*, *nk* are plotted togehter with step delayed by the given dead-time *T*.

When using the same numbers of poles and zeros ($m = n$), the step response of the resulting approximation shows a step at t = 0 which is untypical for a pure time delay. However, it has a magnitude of 1 (0 dB) over all frequencies. In [1], an approximaton having less zeros than poles ($m < n$) is suggested as an alternative approach, resulting in a better step response but decresing magnitude for higher frequencies.

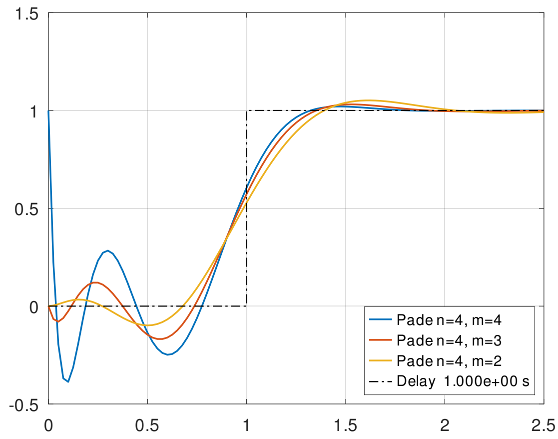
Algorithm based on:

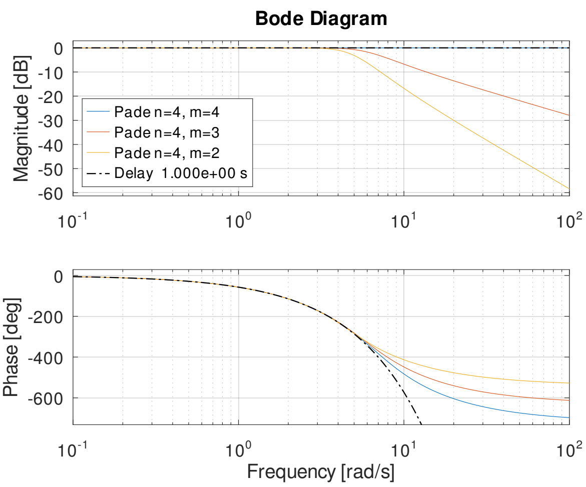
1. Vajta, M. (2000). Some remarks on Pade-approximations, pp53-58, Paper presented at 3rd TEMPUS-INTCOM Symposium on Intelligent Systems in Control and Measurements 2000, Veszprém, Hungary.

See also: @tf/tf, thiran.

Example: 1

```
pade (1,4,4,4,3,4,2)
```





2.3.9 thiran

sys = thiran (*T*, *Ts*) [Function File]

Approximation of continuous-time delay using a discrete-time allpass Thiran filter.

Thiran filters can approximate continuous-time delays that are non-integer multiples of the sampling time (fractional delays). This approximation gives a better matching of the phase shift between the continuous- and the discrete-time system. If there is no fractional part in the delay, then the standard discrete-time delay representation is used.

Inputs

T A continuous-time delay, given in time units (seconds).

Ts The sampling time of the resulting Thiran filter.

Outputs

sys Transfer function model of the resulting filter. The order of the filter is determined automatically.

Example

```
octave:1> sys = thiran (1.33, 0.5)

Transfer function 'sys' from input 'u1' to output ...

      0.003859 z^3 - 0.03947 z^2 + 0.2787 z + 1
 y1:  -----------------------------------------
       z^3 + 0.2787 z^2 - 0.03947 z + 0.003859

Sampling time: 0.5 s
Discrete-time model.
octave:2> sys = thiran (1, 0.5)

Transfer function 'sys' from input 'u1' to output ...
       1
 y1:  ---
      z^2

Sampling time: 0.5 s
Discrete-time model.
```

See also: pade.

2.4 Model Interconnections

2.4.1 @lti/blkdiag

sys = blkdiag (*sys1*, *sys2*, ..., *sysN*) [Function File]

Block-diagonal concatenation of LTI models.

2.4.2 @lti/connect

csys = connect (*sys1*, *sys2*, ..., *sysN*, *in*, *out*) [Function File]
csys = connect (*sys*, *cm*, *in*, *out*) [Function File]

Name-based or index-based interconnections between the outputs and inputs of LTI models.

Inputs

sys1, ..., *sysN*
LTI models to be connected by named-based connection. The properties 'inname' and 'outname' of each model should be set according to the desired input-output connections.

sys LTI model where the outputs are connected to the inputs by index-based connection.

in For name-based interconnections, string or cell of strings containing the names of the inputs to be kept. The names must be part of the properties 'ingroup' or 'inname'. For index-based interconnections, vector containing the indices of the inputs to be kept.

out For name-based interconnections, string or cell of strings containing the names of the outputs to be kept. The names must be part of the properties 'outgroup' or 'outname'. For index-based interconnections, vector containing the indices of the outputs to be kept.

cm Connection matrix (not name-based). Each row of the matrix represents a summing junction. The first column holds the indices of the inputs to be summed with outputs of the subsequent columns. The output indices can be negative, if the output is to be substracted, or zero. For example, the row

```
[2 0 3 -4 0]
```

or

```
[2 -4 3]
```

will sum input u(2) with outputs y(3) and y(4) as

```
u(2) + y(3) - y(4).
```

If several systems are cinnected as in the name-based case, they have to be stacked by `append` before using `connect`.

Outputs

csys Resulting interconnected system with outputs *out* and inputs *in*.

Example

Consider the control loop with reference r, disturbances d1, d2 and an additional output which represents the output of the controller

```
                      d1 --+              d2 --+
                           |                   |
         e  +--------+     v  u  +--------+    v
r --->o---->|  K(s)  |--+->o---->|  G(s)  |--->o--+----> y
      ^ -   +--------+  |        +--------+       |
      |                 |                         |
      |                 +------------------------------> yc
      |                                           |
      +-------------------------------------------+
```

Name-based interconnections:

```
G.inname = 'u';
G.outname = 'y1';
K.inname = 'e';
K.outname = 'yc';
s1 = sumblk ('e = r - y');
s2 = sumblk ('u = yc + d1');
s3 = sumblk ('y = y1 + d2');
in = {'r', 'd1', 'd2'};
out = {'y', 'yc'};
G_cl1 = tf (connect (K, G, s1, s2, s3, in, out))
```

Index-based interconnections (without changing `G` and `K`):

```
G1 = tf (1);   # static gains with r, d1, d2, y as outputs
G_all = append (G1, G1, G1, K, G, G1);
cm = [4, 1, -6; 5, 4, 2; 6, 5, 3];
in = [1, 2, 3];
out = [6, 4];
G_cl2 = connect (G_all, cm, in, out)
```

Index-based interconnections (changing `G` and `K`):

```
[A,B,C,D] = ssdata(K);
K = ss (A, [B -B], C, [D -D]);     # compare s1 above
[A,B,C,D] = ssdata(G);
G = ss (A, [B B 0*B], C, [D D 1]); # compare s2 and s3 above
G_all = append (K, G);
cm = [2, 2; 3, 1];
in = [1, 3, 5];
out = [2, 1];
G_cl3 = tf (connect (G_all, cm, in, out))
```

See also: sumblk, append.

2.4.3 @lti/feedback

sys = `feedback` (*sys1*) [Function File]
sys = `feedback` (*sys1*, `"+"`) [Function File]
sys = `feedback` (*sys1*, *sys2*) [Function File]
sys = `feedback` (*sys1*, *sys2*, `"+"`) [Function File]
sys = `feedback` (*sys1*, *sys2*, *feedin*, *feedout*) [Function File]
sys = `feedback` (*sys1*, *sys2*, *feedin*, *feedout*, `"+"`) [Function File]

Feedback connection of two LTI models.

Inputs

sys1 LTI model of forward transmission. `[p1, m1] = size (sys1)`.

sys2 LTI model of backward transmission. If not specified, an identity matrix of appropriate size is taken.

feedin Vector containing indices of inputs to *sys1* which are involved in the feedback loop. The number of *feedin* indices and outputs of *sys2* must be equal. If not specified, `1:m1` is taken.

feedout Vector containing indices of outputs from *sys1* which are to be connected to *sys2*. The number of *feedout* indices and inputs of *sys2* must be equal. If not specified, `1:p1` is taken.

`"+"` Positive feedback sign. If not specified, `"-"` for a negative feedback interconnection is assumed. *+1* and *-1* are possible as well, but only from the third argument onward due to ambiguity.

Outputs

sys Resulting LTI model.

Block Diagram

```
 u    +         +---------+             y
------>(+)----->|  sys1   |-------+------->
        ^ -     +---------+       |
        |                         |
        |       +---------+       |
        +-------|  sys2   |<------+
                +---------+
```

2.4.4 @lti/lft

sys = lft (*sys1*, *sys2*) [Function File]
sys = lft (*sys1*, *sys2*, *nu*, *ny*) [Function File]

Linear fractional tranformation, also known as Redheffer star product.

Inputs

sys1 Upper LTI model.

sys2 Lower LTI model.

nu The last nu inputs of *sys1* are connected with the first nu outputs of *sys2*. If not specified, `min (m1, p2)` is taken.

ny The last ny outputs of *sys1* are connected with the first ny inputs of *sys2*. If not specified, `min (p1, m2)` is taken.

Outputs

sys Resulting LTI model.

Block Diagram

```
          .............sys..............
          :         +--------+         :
    w1 ------------>|        |------------> z1
          :         |  sys1  |         :
          : u +---->|        |-----+ y :
          :   |     +--------+     |   :          Lower LFT
          :   |                    |   :
          :   |     +--------+     |   :          lft (sys1, sys2)
          :   +-----|  sys2  |<----+   :
          :         +--------+         :
          :............................:

          .............sys..............
          :         +--------+         :
          : u +---->|  sys1  |-----+ y :
          :   |     +--------+     |   :          Upper LFT
          :   |                    |   :
          :   |     +--------+     |   :          lft (sys1, sys2)
          :   +-----|        |<----+   :
          :         |  sys2  |         :
    z2 <------------|        |<------------ w2
          :         +--------+         :
          :............................:
```

```
          .............sys.............
          :         +--------+         :
    w1 ------------>|        |------------> z1
          :         |  sys1  |         :
          : u +---->|        |-----+ y :
          :   |     +--------+     |   :
          :   |                    |   :          lft (sys1, sys2, nu, ny)
          :   |     +--------+     |   :
          :   +-----|        |<----+   :
          :         |  sys2  |         :
    z2 <------------|        |<------------ w2
          :         +--------+         :
          :............................:
```

2.4.5 @lti/mconnect

sys = mconnect (*sys*, *m*) [Function File]
sys = mconnect (*sys*, *m*, *inputs*, *outputs*) [Function File]

Arbitrary interconnections between the inputs and outputs of an LTI model.

Inputs

sys LTI system.

m Connection matrix. Each row belongs to an input and each column represents an output.

inputs Vector of indices of those inputs which are retained. If not specified, all inputs are kept.

outputs Vector of indices of those outputs which are retained. If not specified, all outputs are kept.

Outputs

sys Interconnected system.

Example

```
Solve the system equations of
y(t) = G e(t)
e(t) = u(t) + M y(t)
in order to build
y(t) = H u(t)
The matrix M for a (p-by-m) system G
has m rows and p columns (m-by-p).

Example for a 3x2 system:
u1 = -1*y1 + 5*y2 + 0*y3
u2 = pi*y1 + 0*y2 - 7*y3

    | -1      5      0 |
M = | pi      0      7 |
```

2.4.6 @lti/parallel

sys = parallel (*sys1*, *sys2*) [Function File]

Parallel connection of two LTI systems.

Block Diagram

```
    ..........................
    :      +--------+        :
    :  +-->|  sys1  |---+    :
 u  :  |   +--------+   | +  :  y
-------+                O--------->
    :  |   +--------+   | +  :
    :  +-->|  sys2  |---+    :
    :      +--------+        :
    :.........sys............:

sys = parallel (sys1, sys2)
```

2.4.7 @lti/series

sys = series (*sys1*, *sys2*) [Function File]
sys = series (*sys1*, *sys2*, *outputs1*, *inputs2*) [Function File]

Series connection of two LTI models.

Block Diagram

```
    .....................................
 u  :  +--------+ y1    u2  +--------+  :  y
------>|  sys1  |---------->|  sys2  |------->
    :  +--------+           +--------+  :
    :................sys.................

sys = series (sys1, sys2)

    .....................................
    :                   v2  +--------+  :
    :            ---------->|        |  :  y
    :  +--------+ y1    u2  |  sys2  |------->
 u  :  |        |---------->|        |  :
------>|  sys1  |       z1  +--------+  :
    :  |        |---------->            :
    :  +--------+                       :
    :................sys.................

outputs1 = [1]
inputs2 = [2]
sys = series (sys1, sys2, outputs1, inputs2)
```

2.4.8 append

sys = append (*sys1*, *sys2*, ..., *sysN*) [Function File]

Group LTI models by appending their inputs and outputs.

2.4.9 sumblk

S = sumblk (*formula*) [Function File]
S = sumblk (*formula*, *n*) [Function File]

Create summing junction *S* from string *formula* for name-based interconnections.

Inputs

formula String containing the formula of the summing junction, e.g. `e = r - y + d` *n* Signal size. Default value is 1.

Outputs

S State-space model of the summing junction.

Example

```
octave:1> S = sumblk ('e = r - y + d')

S.d =
       r   y   d
   e   1  -1   1

Static gain.
octave:2> S = sumblk ('e = r - y + d', 2)

S.d =
       r1  r2  y1  y2  d1  d2
   e1   1   0  -1   0   1   0
   e2   0   1   0  -1   0   1

Static gain.
```

See also: @lti/connect.

2.5 Model Characteristics

2.5.1 @lti/dcgain

k = dcgain (*sys*) [Function File]

Compute the DC gain of LTI system.

Inputs

sys LTI system created by tf(), ss(), dss(), etc.

Outputs

k DC gain matrix. For a system with m inputs and p outputs, the array *k* has dimensions [p, m].

Transfer function for a continuous state space system (A,B,C,D) G(s) = C * inv(s*I - A) * B + D

DC Gain: evaluate G(s) as s -> 0: k = C * inv(-A) * B + D

Transfer function for a discrete state space system (A,B,C,D,T) G(z) = C * inv(z*I - A) * B + D

DC Gain: evaluate G(z) as z -> 1: k = C * inv(I-A) * B + D

Example

```
G = Transfer function 'G' from input 'u1' to output ...

                  1
y1:  ---------------------
     s^3 + 2 s^2 + 3 s + 4


octave:1> K = dcgain(G)
```

```
K =  0.25000
```

See also: @lti/freqresp, @tf/tf, @ss/ss, dss.

2.5.2 @lti/isct

bool = isct (*sys*) [Function File]

Determine whether LTI model is a continuous-time system.

Inputs

sys LTI system.

Outputs

bool = 0 *sys* is a discrete-time system.

bool = 1 *sys* is a continuous-time system or a static gain.

2.5.3 @lti/isdt

bool = isdt (*sys*) [Function File]

Determine whether LTI model is a discrete-time system.

Inputs

sys LTI system.

Outputs

bool = 0 *sys* is a continuous-time system.

bool = 1 *sys* is a discrete-time system or a static gain.

2.5.4 @lti/isminimumphase

bool = isminimumphase (*sys*) [Function File]

bool = isminimumphase (*sys*, *tol*) [Function File]

Determine whether LTI system has asymptotically stable zero dynamics. According to the definition of Byrnes/Isidori [1], the zeros of a minimum-phase system must be strictly inside the left complex half-plane (continuous-time case) or inside the unit circle (discrete-time case). Note that the poles are not tested.

M. Zeitz [2] discusses the inconsistent definitions of the minimum-phase property in a German paper. The abstract in English states the following [2]:

Originally, the minimum phase property has been defined by H. W. Bode [3] in order to characterize the unique relationship between gain and phase of the frequency response. With regard to the design of digital filters, another definition of minimum phase is used and a filter is said to be minimum phase if both the filter and its inverse are asymptotically stable. Finally, systems with asymptotically stable zero dynamics are named as minimum phase by C. I. Byrnes and A. Isidori [1]. Due to the inconsistent definitions, avoiding the minimum phase property for control purposes is advocated and the well-established criteria of Hurwitz or Ljapunow to describe the stability of filters and zero dynamics are recommended.

Inputs

sys LTI system.

tol Optional tolerance. *tol* must be a real-valued, non-negative scalar. Default value is 0.

Outputs

bool True if the system is minimum-phase and false otherwise.

```
real (z) < -tol*(1 + abs (z))    continuous-time
abs (z) < 1 - tol                discrete-time
```

References

1. Byrnes, C.I. and Isidori, A. *A Frequency Domain Philosophy for Nonlinear Systems.* IEEE Conf. Dec. Contr. 23, pp. 1569–1573, 1984.
2. Zeitz, M. *Minimum phase – no relevant property of automatic control!.* at – Automatisierungstechnik. Volume 62, Issue 1, pp. 3–10, 2014.
3. Bode, H.W. *Network Analysis and Feedback Amplifier Design.* D. Van Nostrand Company, pp. 312-318, 1945. pp. 341-351, 1992.

2.5.5 @lti/issiso

bool = issiso (*sys*) [Function File]

Determine whether LTI model is single-input/single-output (SISO).

2.5.6 @lti/isstable

bool = isstable (*sys*) [Function File]
bool = isstable (*sys*, *tol*) [Function File]

Determine whether LTI system is stable.

Inputs

sys LTI system.

tol Optional tolerance for stability. *tol* must be a real-valued, non-negative scalar. Default value is 0.

Outputs

bool True if the system is stable and false otherwise.

```
real (p) < -tol*(1 + abs (p))    continuous-time
abs (p) < 1 - tol                discrete-time
```

2.5.7 @lti/norm

gain = norm (*sys*, *2*) [Function File]
[*gain*, *wpeak*] = norm (*sys*, *inf*) [Function File]
[*gain*, *wpeak*] = norm (*sys*, *inf*, *tol*) [Function File]

Return H-2 or L-inf norm of LTI model.

Algorithm

Uses SLICOT AB13BD and AB13DD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.5.8 @lti/pole

p = pole (*sys*) [Function File]

Compute poles of LTI system.

Inputs

sys LTI model.

Outputs

p Poles of *sys*.

Algorithm
For (descriptor) state-space models and system/state matrices, `pole` relies on Octave's `eig`. For SISO transfer functions, `pole` uses Octave's `roots`. MIMO transfer functions are converted to a *minimal* state-space representation for the computation of the poles.

2.5.9 @lti/size

`nvec = size (sys)` [Function File]
`n = size (sys, dim)` [Function File]
`[p, m] = size (sys)` [Function File]

LTI model size, i.e. number of outputs and inputs.

Inputs

sys LTI system.

dim If given a second argument, `size` will return the size of the corresponding dimension.

Outputs

nvec Row vector. The first element is the number of outputs (rows) and the second element the number of inputs (columns).

n Scalar value. The size of the dimension *dim*.

p Number of outputs.

m Number of inputs.

2.5.10 @lti/zero

`z = zero (sys)` [Function File]
`z = zero (sys, type)` [Function File]
`[z, k, info] = zero (sys)` [Function File]

Compute zeros and gain of LTI model. By default, `zero` computes the invariant zeros, also known as Smith zeros. Alternatively, when called with a second input argument, `zero` can also compute the system zeros, transmission zeros, input decoupling zeros and output decoupling zeros. See paper [1] for an explanation of the various zero flavors as well as for further details.

Inputs

sys LTI model.

type String specifying the type of zeros:

'system', 's'
Compute the system zeros. The system zeros include in all cases (square, non-square, degenerate or non-degenerate system) all transmission and decoupling zeros.

'invariant', 'inv'
Compute invariant zeros. Default selection.

'transmission', 't'
Compute transmission zeros. Transmission zeros are a subset of the invariant zeros. The transmission zeros are the zeros of the Smith-McMillan form of the transfer function matrix.

'input', 'inp', 'id'
: Compute input decoupling zeros. The input decoupling zeros are also known as the uncontrollable eigenvalues of the pair (A,B).

'output', 'o', 'od'
: Compute output decoupling zeros. The output decoupling zeros are also known as the unobservable eigenvalues of the pair (A,C).

Outputs

z Depending on argument *type*, *z* contains the invariant (default), system, transmission, input decoupling or output decoupling zeros of *sys* as defined in [1].

k Gain of SISO system *sys*. For MIMO systems, an empty matrix `[]` is returned.

info Struct containing additional information. For details, see the documentation of SLICOT routines AB08ND and AG08BD.

info.rank The normal rank of the transfer function matrix (regular state-space models) or of the system pencil (descriptor state-space models).

info.infz Contains information on the infinite elementary divisors as follows: the system has info.infz(i) infinite elementary divisors of degree i, where i=1,2,...,length(info.infz).

info.kronr Right Kronecker (column) indices.

info.kronl Left Kronecker (row) indices.

Examples

```
[z, k, info] = zero (sys)        # invariant zeros
z = zero (sys, 'system')         # system zeros
z = zero (sys, 'invariant')      # invariant zeros
z = zero (sys, 'transmission')   # transmission zeros
z = zero (sys, 'output')         # output decoupling zeros
z = zero (sys, 'input')          # input decoupling zeros
```

Algorithm

For (descriptor) state-space models, `zero` uses SLICOT AB08ND and AG08BD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer). For SISO transfer functions, `zero` uses Octave's `roots`. MIMO transfer functions are converted to a *minimal* state-space representation for the computation of the zeros.

References

1. MacFarlane, A. and Karcanias, N. *Poles and zeros of linear multivariable systems: a survey of the algebraic, geometric and complex-variable theory*. Int. J. Control, vol. 24, pp. 33-74, 1976.

2. Rosenbrock, H.H. *Correction to 'The zeros of a system'*. Int. J. Control, vol. 20, no. 3, pp. 525-527, 1974.

3. Svaricek, F. *Computation of the structural invariants of linear multivariable systems with an extended version of the program ZEROS*. Systems & Control Letters, vol. 6, pp. 261-266, 1985.

4. Emami-Naeini, A. and Van Dooren, P. *Computation of zeros of linear multivariable systems*. Automatica, vol. 26, pp. 415-430, 1982.

2.5.11 compreal

`[csys, T] = compreal (sys)` [Function File]
`[csys, T] = compreal (sys, realization)` [Function File]

Returns the state-space companion controllable ors observable form of the input system as well as the transformation matrix.

Inputs

sys LTI model.

realization The *realization* argument selects the companion form:

- `'c'` Controllable companion form, the default if *realization* is omitted.
- `'o'` Observable companion form.

Outputs

csys State-space model in companian form.

T Transformation matix.

Given the state vector

$$x(t)$$

We look for a transformation matrix such that

$$x(t) = Tz(t)$$

Where T is our transformation matrix and $z(t)$ is our new state vector. Our new state-space is given by the following relationships:

$$A_z = T^{-1}AT$$
$$B_z = T^{-1}B$$
$$C_z = CT$$
$$D_z = D$$

Alorithm based on

1. T. Kailath, Linear systems. Prentice-Hall Englewood Cliffs, NJ, 1980. ISBM 0-13-536961-4. Eequation (15a) and (15b)

2.5.12 ctrb

`co = ctrb (sys)` [Function File]
`co = ctrb (a, b)` [Function File]

Return controllability matrix.

Inputs

sys LTI model.

a State matrix (n-by-n).

b Input matrix (n-by-m).

Outputs

co Controllability matrix.

Equation

$$C_o = [\, B \quad AB \quad A^2B \quad \ldots \quad A^{n-1}B \,]$$

2.5.13 ctrbf

`[sysbar, T, K] = ctrbf (sys)` [Function File]
`[sysbar, T, K] = ctrbf (sys, tol)` [Function File]
`[Abar, Bbar, Cbar, T, K] = ctrbf (A, B, C)` [Function File]
`[Abar, Bbar, Cbar, T, K] = ctrbf (A, B, C, TOL)` [Function File]

If Co=ctrb(A,B) has rank r <= n = SIZE(A,1), then there is a similarity transformation Tc such that Tc = [t1 t2] where t1 is the controllable subspace and t2 is orthogonal to t1

```
Abar = Tc \ A * Tc ,  Bbar = Tc \ B ,  Cbar = C * Tc
```

and the transformed system has the form

```
       | Ac    A12|           | Bc |
Abar = |----------|,   Bbar = | ---|,  Cbar = [Cc | Cnc].
       | 0     Anc|           |  0 |
```

where (Ac,Bc) is controllable, and Cc(sI-Ac)^(-1)Bc = C(sI-A)^(-1)B. and the system is stabilizable if Anc has no eigenvalues in the right half plane. The last output K is a vector of length n containing the number of controllable states.

2.5.14 damp

`damp(sys)` [Function File]
`[Wn, zeta] = damp(sys)` [Function File]
`[Wn, zeta, P] = damp(sys)` [Function File]

Calculate natural frequencies, damping ratios and poles.

If no output is specified, display overview table containing poles, magnitude (if *sys* is a discrete-time model), damping ratios, natural frequencies and time constants.

Inputs

sys LTI model.

Outputs

Wn Natural frequencies of each pole of *sys* (in increasing order). The frequency unit is rad/s (radians per second). If *sys* is a discrete-time model with specified sample time, *Wn* contains the natural frequencies of the equivalent continuous-time poles (see Algorithms). If *sys* has an unspecified sample time (*tsam* = -1), *tsam* = 1 is used to calculate *Wn*.

zeta Damping ratios of each pole of *sys* (in the same order as *Wn*). If *sys* is a discrete-time model with specified sample time, *zeta* contains the damping ratios of the equivalent continuous-time poles (see Algorithms). If *sys* has an unspecified sample time (*tsam* = -1), *tsam* = 1 is used to calculate *zeta*.

P Poles of *sys* (in the same order as *Wn*).

Algorithm

Pole Poles s (or z for discrete-time models) are calculated via `pole` and resorted in order of increasing natural frequency.

Equivalent continuous-time pole
s = log (z) / sys.tsam (discrete-time models only)

Magnitude
mag = abs (z) (discrete-time models only)

Natural Frequency
Wn = abs (s)

Damping ratio
zeta = -cos (arg (s))

Time constant
tau = 1 / (Wn * zeta)

See also: dsort, esort, @lti/pole, pzmap, @lti/zero.

2.5.15 dsort

`s = dsort(p)` [Function File]
`[s, ndx] = dsort(p)` [Function File]

Sort discrete-time poles by magnitude (in decreasing order).

Inputs

p Input vector containing discrete-time poles.

Outputs

s Vector with sorted poles.

ndx Vector containing the indices used in the sort.

See also: esort, @lti/pole, pzmap, @lti/zero.

2.5.16 esort

`s = esort(p)` [Function File]
`[s, ndx] = esort(p)` [Function File]

Sort continuous-time poles by real part (in decreasing order).

Inputs

p Input vector containing continuous-time poles.

Outputs

s Vector with sorted eigenvalues.

ndx Vector containing the indices used in the sort.

See also: dsort, eig, @lti/pole, pzmap, sort, @lti/zero.

2.5.17 gram

`W = gram (sys, mode)` [Function File]
`Wc = gram (a, b)` [Function File]

`gram (sys, "c")` returns the controllability gramian of the (continuous- or discrete-time) system *sys*. `gram (sys, "o")` returns the observability gramian of the (continuous- or discrete-time) system *sys*. `gram (a, b)` returns the controllability gramian *Wc* of the continuous-time system $\dot{x} = Ax + Bu$ i.e., *Wc* satisfies $AWc + mWc^T + BB^T = 0$.

2.5.18 hsvd

`hsv = hsvd (sys)` [Function File]
`hsv = hsvd (sys, "offset", offset)` [Function File]
`hsv = hsvd (sys, "alpha", alpha)` [Function File]

Hankel singular values of the stable part of an LTI model. If no output arguments are given, the Hankel singular values are displayed in a plot.

Algorithm
Uses SLICOT AB13AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.5.19 isctrb

[*bool*, *ncon*] = isctrb (*sys*) [Function File]
[*bool*, *ncon*] = isctrb (*sys*, *tol*) [Function File]
[*bool*, *ncon*] = isctrb (*a*, *b*) [Function File]
[*bool*, *ncon*] = isctrb (*a*, *b*, *e*) [Function File]
[*bool*, *ncon*] = isctrb (*a*, *b*, [], *tol*) [Function File]
[*bool*, *ncon*] = isctrb (*a*, *b*, *e*, *tol*) [Function File]

Logical check for system controllability. For numerical reasons, `isctrb (sys)` should be used instead of `rank (ctrb (sys))`.

Inputs

sys LTI model. Descriptor state-space models are possible. If *sys* is not a state-space model, it is converted to a minimal state-space realization, so beware of pole-zero cancellations which may lead to wrong results!

a State matrix (n-by-n).

b Input matrix (n-by-m).

e Descriptor matrix (n-by-n). If *e* is empty `[]` or not specified, an identity matrix is assumed.

tol Optional roundoff parameter. Default value is 0.

Outputs

bool = 0 System is not controllable.

bool = 1 System is controllable.

ncon Number of controllable states.

Algorithm
Uses SLICOT AB01OD and TG01HD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: isobsv.

2.5.20 isdetectable

bool = isdetectable (*sys*) [Function File]
bool = isdetectable (*sys*, *tol*) [Function File]
bool = isdetectable (*a*, *c*) [Function File]
bool = isdetectable (*a*, *c*, *e*) [Function File]
bool = isdetectable (*a*, *c*, [], *tol*) [Function File]
bool = isdetectable (*a*, *c*, *e*, *tol*) [Function File]
bool = isdetectable (*a*, *c*, [], [], *dflg*) [Function File]
bool = isdetectable (*a*, *c*, *e*, [], *dflg*) [Function File]
bool = isdetectable (*a*, *c*, [], *tol*, *dflg*) [Function File]
bool = isdetectable (*a*, *c*, *e*, *tol*, *dflg*) [Function File]

Logical test for system detectability. All unstable modes must be observable or all unobservable states must be stable.

Inputs

sys LTI system.

a State transition matrix.

c Measurement matrix.

e Descriptor matrix. If *e* is empty `[]` or not specified, an identity matrix is assumed.

tol Optional tolerance for stability. Default value is 0.

dflg = 0 Matrices (*a*, *c*) are part of a continuous-time system. Default Value.

dflg = 1 Matrices (*a*, *c*) are part of a discrete-time system.

Outputs

bool = 0 System is not detectable.

bool = 1 System is detectable.

Algorithm
Uses SLICOT AB01OD and TG01HD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See `isstabilizable` for description of computational method. **See also:** isstabilizable, @lti/isstable, isctrb, isobsv.

2.5.21 isobsv

```
[bool, nobs] = isobsv (sys)                [Function File]
[bool, nobs] = isobsv (sys, tol)           [Function File]
[bool, nobs] = isobsv (a, c)               [Function File]
[bool, nobs] = isobsv (a, c, e)            [Function File]
[bool, nobs] = isobsv (a, c, [], tol)      [Function File]
[bool, nobs] = isobsv (a, c, e, tol)       [Function File]
```

Logical check for system observability. For numerical reasons, `isobsv (sys)` should be used instead of `rank (obsv (sys))`.

Inputs

sys LTI model. Descriptor state-space models are possible.

a State matrix (n-by-n).

c Measurement matrix (p-by-n).

e Descriptor matrix (n-by-n). If *e* is empty `[]` or not specified, an identity matrix is assumed.

tol Optional roundoff parameter. Default value is 0.

Outputs

bool = 0 System is not observable.

bool = 1 System is observable.

nobs Number of observable states.

Algorithm
Uses SLICOT AB01OD and TG01HD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: isctrb.

2.5.22 isstabilizable

bool = isstabilizable (*sys*) [Function File]
bool = isstabilizable (*sys*, *tol*) [Function File]
bool = isstabilizable (*a*, *b*) [Function File]
bool = isstabilizable (*a*, *b*, *e*) [Function File]
bool = isstabilizable (*a*, *b*, [], *tol*) [Function File]
bool = isstabilizable (*a*, *b*, *e*, *tol*) [Function File]
bool = isstabilizable (*a*, *b*, [], [], *dflg*) [Function File]
bool = isstabilizable (*a*, *b*, *e*, [], *dflg*) [Function File]
bool = isstabilizable (*a*, *b*, [], *tol*, *dflg*) [Function File]
bool = isstabilizable (*a*, *b*, *e*, *tol*, *dflg*) [Function File]

Logical check for system stabilizability. All unstable modes must be controllable or all uncontrollable states must be stable.

Inputs

sys LTI system. If *sys* is not a state-space system, it is converted to a minimal state-space realization, so beware of pole-zero cancellations which may lead to wrong results!

a State transition matrix.

b Input matrix.

e Descriptor matrix. If *e* is empty `[]` or not specified, an identity matrix is assumed.

tol Optional tolerance for stability. Default value is 0.

dflg = 0 Matrices (*a*, *b*) are part of a continuous-time system. Default Value.

dflg = 1 Matrices (*a*, *b*) are part of a discrete-time system.

Outputs

bool = 0 System is not stabilizable.

bool = 1 System is stabilizable.

Algorithm

Uses SLICOT AB01OD and TG01HD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

```
* Calculate staircase form (SLICOT AB01OD)
* Extract unobservable part of state transition matrix
* Calculate eigenvalues of unobservable part
* Check whether
  real (ev) < -tol*(1 + abs (ev))   continuous-time
  abs (ev) < 1 - tol                discrete-time
```

See also: isdetectable, @lti/isstable, isctrb, isobsv.

2.5.23 obsv

ob = obsv (*sys*) [Function File]
ob = obsv (*a*, *c*) [Function File]

Return observability matrix.

Inputs

sys LTI model.

a State matrix (n-by-n).

c Measurement matrix (p-by-n).

Outputs

ob Observability matrix.

Equation

$$O_b = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

2.5.24 obsvf

`[sysbar, T, K] = obsvf (sys)` [Function File]
`[sysbar, T, K] = obsvf (sys, tol)` [Function File]
`[Abar, Bbar, Cbar, T, K] = obsvf (A, B, C)` [Function File]
`[Abar, Bbar, Cbar, T, K] = obsvf (A, B, C, TOL)` [Function File]

If Ob=obsv(A,C) has rank r <= n = SIZE(A,1), then there is a similarity transformation Tc such that To = [t1;t2] where t1 is c and t2 is orthogonal to t1

```
Abar = To \ A * To ,  Bbar = To \ B ,  Cbar = C * To
```

and the transformed system has the form

```
       | Ao     0 |           | Bo  |
Abar = |----------|,   Bbar = | --- |,  Cbar = [Co | 0 ].
       | A21   Ano|           | Bno |
```

where (Ao,Bo) is observable, and Co(sI-Ao)^(-1)Bo = C(sI-A)^(-1)B. And system is detectable if Ano has no eigenvalues in the right half plane. The last output K is a vector of length n containing the number of observable states.

2.5.25 pzmap

`pzmap (sys)` [Function File]
`pzmap (sys1, sys2, ..., sysN)` [Function File]
`pzmap (sys1, 'style1', ..., sysN, 'styleN')` [Function File]
`[p, z] = pzmap (sys)` [Function File]

Plot the poles and zeros of an LTI system in the complex plane. If no output arguments are given, the result is plotted on the screen. Otherwise, the poles and zeros are computed and returned. Note that only one system is processed when output arguments are given.

Inputs

sys, sys1, ...
LTI model(s).

'style' Color, e.g. 'r' for a red. See `help plot` for details. Marker or line styles are ignored as poles and zeros have the fixed marker types 'x' and 'o' repectively.

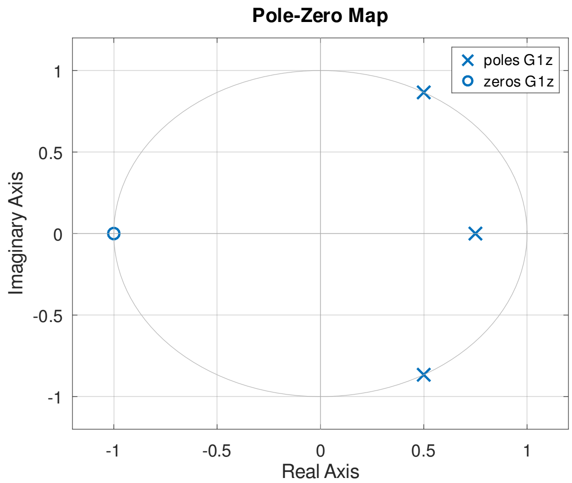
Outputs

p Poles of *sys*.

z Invariant zeros of *sys*.

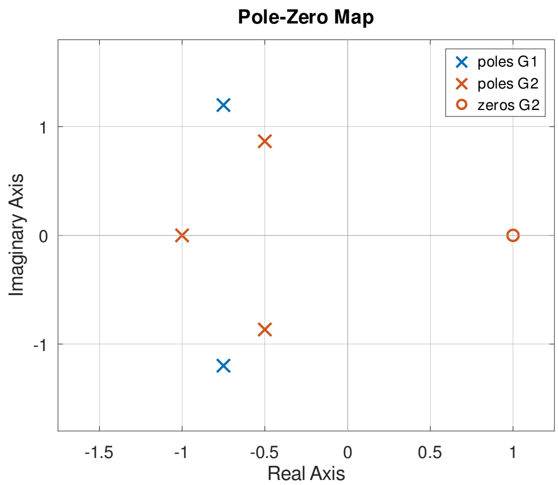
Example: 1

```
z = tf('z',1);
G1z = (z+1)/(z-0.75)/(z^2-1*z+1);
pzmap(G1z);
```



Example: 2

```
s = tf('s');
G1 = 1/(2*s^2+3*s+4);
G2 = (1-s)/(1+s)/(s^2+s+1);
pzmap(G1,G2);
```



2.6 Model Simplification

2.6.1 @lti/minreal

sys = `minreal` (*sys*) [Function File]
sys = `minreal` (*sys*, *tol*) [Function File]

Minimal realization or zero-pole cancellation of LTI models.

2.6.2 @lti/sminreal

sys = `sminreal` (*sys*) [Function File]
sys = `sminreal` (*sys*, *tol*) [Function File]

Perform state-space model reduction based on structure. Remove states which have no influence on the input-output behaviour. The physical meaning of the states is retained.

Inputs

sys State-space model.

tol Optional tolerance for controllability and observability. Entries of the state-space matrices whose moduli are less or equal to *tol* are assumed to be zero. Default value is 0.

Outputs

sys Reduced state-space model.

See also: @lti/minreal.

2.7 Time Domain Analysis

2.7.1 covar

`[p, q] = covar (sys, w)` [Function File]

Return the steady-state covariance.

Inputs

sys LTI model.

w Intensity of Gaussian white noise inputs which drive *sys*.

Outputs

p Output covariance.

q State covariance.

See also: lyap, dlyap.

2.7.2 gensig

`[u, t] = gensig (sigtype, tau)` [Function File]
`[u, t] = gensig (sigtype, tau, tfinal)` [Function File]
`[u, t] = gensig (sigtype, tau, tfinal, tsam)` [Function File]

Generate periodic signal. Useful in combination with lsim.

Inputs

sigtype = "*sin*"
Sine wave.

sigtype = "*cos*"
Cosine wave.

sigtype = "*square*"
Square wave.

sigtype = "*pulse*"
Periodic pulse.

tau Duration of one period in seconds.

tfinal Optional duration of the signal in seconds. Default duration is 5 periods.

tsam Optional sampling time in seconds. Default spacing is tau/64.

Outputs

u Vector of signal values.

t Time vector of the signal.

See also: lsim.

2.7.3 imp_invar

`[b_out, a_out] = imp_invar (b, a, fs, tol)` [Function File]
`[b_out, a_out] = imp_invar (b, a, fs)` [Function File]
`[b_out, a_out] = imp_invar (b, a)` [Function File]
`[sys_out] = imp_invar (b, a, fs. tol)` [Function File]
`[sys_out] = imp_invar (sys_in, fs, tol)` [Function File]

Converts analog filter with coefficients *b* and *a* and/or *sys_in* to digital, conserving impulse response.

MIMO systems are only supported with *sys_in* as input argument.

Inputs

b Numerator coefficients of continuous-time LTI system.

a Denominator coefficients of continuous-time LTI system.

fs Sampling frequency. If *fs* is not specified, or is an empty vector, it defaults to 1Hz. tol Tolerance of the internally used function minreal for canceling identical poles and zeros. If *tol* is not specified, it defaults to 0.0001 (0.1%).

sys_in System definition of the continuous-time LTI system. This can also be a MIMO system.

Outputs

b_out Numerator coefficients of the discrete-time impulse invariant LTI system.

a_out Denominator coefficients of the discrete-time impulse invariant LTI system.

sys_out Discrete-time impulse invaraiant LTI system. If *sys_in* is given as state space representation, *sys_out* is also returned in state space, otherwise as transfer function.

Algorithm

The step equivalent discretization of G(s) (zoh) results in G_zoh(z) = (z-1)/z * Z{G(s)/s} where Z{} is the z-transformation. The transfer function of the impulse equivalent discretization is given by T*Z{G(s)}. Therefore, the zoh discretizaiton method for s*G(s) multipled by T*z/(z-1) leads to the desired result.

Remark

For the impulse response of a discrete-time system, the input sequence {1/T,0,0,0,...} and not the unit impulse is considered. For this reason, the factor T is required for the impulse invaraint discretizaiton (see Algorithm).

See also: @lti/c2d.

2.7.4 impulse

`impulse (sys)` [Function File]
`impulse (sys1, sys2, ..., sysN)` [Function File]
`impulse (sys1, 'style1', ..., sysN, 'styleN')` [Function File]
`impulse (sys1, ..., t)` [Function File]
`impulse (sys1, ..., tfinal)` [Function File]
`impulse (sys1, ..., tfinal, dt)` [Function File]
`[y, t, x] = impulse (sys)` [Function File]
`[y, t, x] = impulse (sys, t)` [Function File]
`[y, t, x] = impulse (sys, tfinal)` [Function File]
`[y, t, x] = impulse (sys, tfinal, dt)` [Function File]

Impulse response of LTI system. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI model.

t Time vector. Should be evenly spaced. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

tfinal Optional simulation horizon. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

dt Optional sampling time. Be sure to choose it small enough to capture transient phenomena. If not specified, it is calculated by the poles of the system.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

y Output response array. Has as many rows as time samples (length of t) and as many columns as outputs.

t Time row vector.

x State trajectories array. Has `length (t)` rows and as many columns as states.

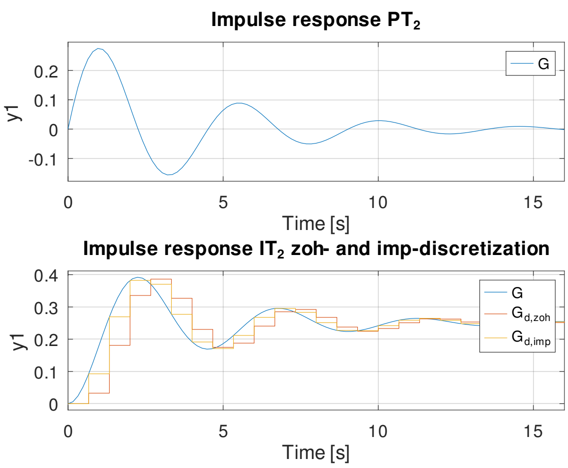
Remark

For the impulse response of a discrete-time system, the input sequence {1/T,0,0,0,...} and not the unit impulse is considered.

See also: initial, lsim, step.

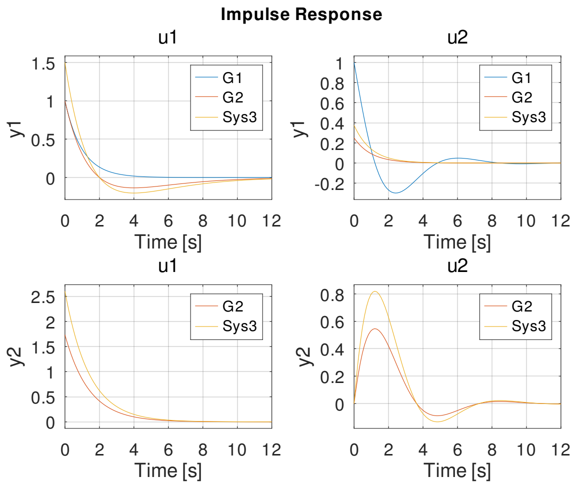
Example: 1

```
clf;
subplot (2,1,1)
G = tf (1,[2 1 4]);
impulse (G,16);
title ("Impulse response PT_2");
subplot (2,1,2)
G = tf (1,[2 1 4 0 ]);
T = 2/3;
Gdzoh = c2d(G, T);
Gdimp = c2d(G, T, 'impulse');
impulse(G, Gdzoh, ';G_{d,zoh};', Gdimp, ';G_{d,imp};', 16);
title ("Impulse response IT_2 zoh- and imp-discretization");
```



Example: 2

```
clf;
warning ("off", "Control:convert-to-state-space");
G1 = tf({[1],[1 0]},{[1 1],[1 1 1]});
G2 = tf  ({[1 0],[0.25];[-2 1],[1]},{[1 1 0.25],[1 1];[1.4 1],[1 1 1]});
impulse (G1,G2,1.5*G2,12);
warning ("on", "Control:convert-to-state-space");
```



2.7.5 initial

`initial` (*sys*, *x0*) [Function File]
`initial` (*sys1*, *sys2*, ..., *sysN*, *x0*) [Function File]
`initial` (*sys1*, *'style1'*, ..., *sysN*, *'styleN'*, *x0*) [Function File]
`initial` (*sys1*, ..., *x0*, *t*) [Function File]
`initial` (*sys1*, ..., *x0*, *tfinal*) [Function File]
`initial` (*sys1*, ..., *x0*, *tfinal*, *dt*) [Function File]
[*y*, *t*, *x*] = `initial` (*sys*, *x0*) [Function File]
[*y*, *t*, *x*] = `initial` (*sys*, *x0*, *t*) [Function File]
[*y*, *t*, *x*] = `initial` (*sys*, *x0*, *tfinal*) [Function File]
[*y*, *t*, *x*] = `initial` (*sys*, *x0*, *tfinal*, *dt*) [Function File]

Initial condition response of a state-space model. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI system in state-space representation.

x0 Vector of initial conditions for each state.

t Optional time vector. Should be evenly spaced. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

tfinal Optional simulation horizon. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

dt Optional sampling time. Be sure to choose it small enough to capture transient phenomena. If not specified, it is calculated by the poles of the system.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

y Output response array. Has as many rows as time samples (length of t) and as many columns as outputs.

t Time row vector.

x State trajectories array. Has `length (t)` rows and as many columns as states.

Example

Consider a continuous- or a discrete-time system of the form

Continuous Time:

$$\dot{x}(t) = A\,x(t) + B\,u(t), \quad x(0) = x_0, \quad y(t) = C\,x(t) + D\,u(t)$$

Discrete Time:

$$x(k+1) = A\,x(k) + B\,u(k), \quad x(0) = x_0, \quad y(k) = C\,x(k) + D\,u(K)$$

The dynamic behavior of the system for $u = 0$ and only driven by the initial system state $x(0)$ is given by

```
sys = ss (A, B, C, D);
initial (sys, x0);
```

Remark

For a SISO input-output model G and initial values for the output y and its derivatives up to order $n-1$ the corresponding state space represetaiton is computed by:

```
[A,b,c,d] = ssdata (G);
T = obsv (A, c);
G_ss = ss2ss (ss (G), T);
initial (G_ss, x0);
```

Note that, in general, the states of $G_s s$ are only equal to the output y and its first $n-1$ time derivaties if $u = 0$, which is the case for the initial condition response.

See also: impulse, lsim, step.

2.7.6 lsim

`lsim (`*sys*`, `*u*`)` [Function File]
`lsim (`*sys1*`, `*sys2*`, ..., `*sysN*`, `*u*`)` [Function File]
`lsim (`*sys1*`, `*style1*`, ..., `*sysN*`, `*styleN*`, `*u*`)` [Function File]
`lsim (`*sys1*`, ..., `*u*`, `*t*`)` [Function File]
`lsim (`*sys1*`, ..., `*u*`, `*t*`, `*x0*`)` [Function File]
`[`*y*`, `*t*`, `*x*`] = lsim (`*sys*`, `*u*`)` [Function File]
`[`*y*`, `*t*`, `*x*`] = lsim (`*sys*`, `*u*`, `*t*`)` [Function File]
`[`*y*`, `*t*`, `*x*`] = lsim (`*sys*`, `*u*`, `*t*`, `*x0*`)` [Function File]
`[...] = lsim (..., `*method*`)` [Function File]

Simulate LTI model response to arbitrary inputs. If no output arguments are given, the system response is plotted on the screen.

Inputs

sys LTI model. System must be proper, i.e. it must not have more zeros than poles.

u Vector or array of input signal. Needs `length(t)` rows and as many columns as there are inputs. If *sys* is a single-input system, row vectors *u* of length `length(t)` are accepted as well.

t Time vector. Should be evenly spaced. If *sys* is a continuous-time system and *t* is a real scalar, *sys* is discretized with sampling time `tsam = t/(rows(u)-1)`. If *sys* is a discrete-time system and *t* is not specified, vector *t* is assumed to be `0 : tsam : tsam*(rows(u)-1)`.

x0 Vector of initial conditions for each state of a system in state space. If not specified, a zero vector is assumed. Note: A vector *x0* provided for an input-output system representation is ignored and a zero vector of initial conditions is used instead because the internally used state space representaton does generally not match the one assumed for *x0*. For a simulation of an input-output model with initial conditions for the output y and its time-derivatives, see remarks below.

style Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

method Method that is used to discretize a continuous-time system for the simulation. See Section 2.3.1 [@lti/c2d], page 13 for possible methods. If *method* is not provided, the default is 'foh'.

Outputs

y Output response array. Has as many rows as time samples (length of t) and as many columns as outputs.

t Time row vector. It is always evenly spaced.

x State trajectories array. Has `length (t)` rows and as many columns as states.

Remarks

- For the simulation, continuous-time systems are discretized using the selected method *method* or the default first-order-hold method (foh), see Section 2.3.1 [@lti/c2d], page 13 for details.
- For a SISO input-output model G and initial values for the output y and its derivatives up to order $n-1$ the corresponding state space represetaiton is computed by:

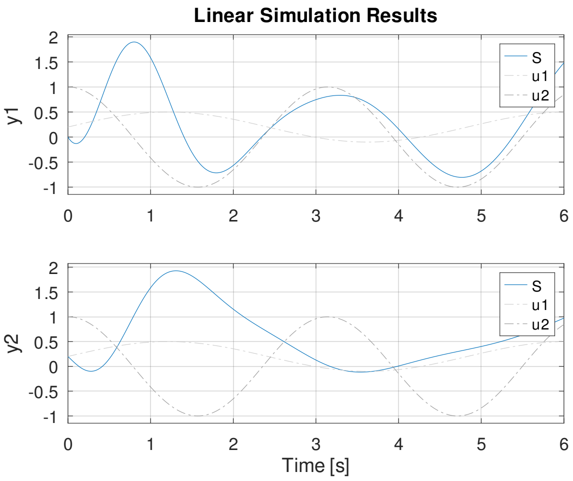
```
[A,b,c,d] = ssdata (G);
T = obsv (A, c);
G_ss = ss2ss (ss (G), T);
initial (G_ss, x0);
```

  Note that, in general, the states of $G_s s$ are only equal to the output y and its first $n-1$ time derivaties if $u = 0$, which is the case for the initial conditions immediately before $t = 0$.

See also: impulse, initial, step.

Example: 1

```
clf;
A = [-3   0   0;
       0  -2   1;
      10 -17   0];
B = [4  0;
      0 -1;
      0 -1];
C = [0 0 1;
      1 2 0];
D = [ 0 0;
       0 0 ];
S = ss(A,B,C,D);
t = 0:0.01:6;
u = [ 0.2+0.3*sin(1.3*t') , cos(2*t') ];
x0 = [0 0.1 0];
lsim(S, u, t, x0);
```



2.7.7 ramp

ramp (*sys*) [Function File]
ramp (*sys1*, *sys2*, ..., *sysN*) [Function File]
ramp (*sys1*, *'style1'*, ..., *sysN*, *'styleN'*) [Function File]
ramp (*sys1*, ..., *t*) [Function File]
ramp (*sys1*, ..., *tfinal*) [Function File]
ramp (*sys1*, ..., *tfinal*, *dt*) [Function File]
[*y*, *t*, *x*] = ramp (*sys*) [Function File]
[*y*, *t*, *x*] = ramp (*sys*, *t*) [Function File]
[*y*, *t*, *x*] = ramp (*sys*, *tfinal*) [Function File]
[*y*, *t*, *x*] = ramp (*sys*, *tfinal*, *dt*) [Function File]

Ramp response of LTI system. If no output arguments are given, the response is printed on the screen.

$$r(t) = t \cdot h(t)$$

Inputs

sys LTI model.

t Time vector. Should be evenly spaced. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

tfinal Optional simulation horizon. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

dt Optional sampling time. Be sure to choose it small enough to capture transient phenomena. If not specified, it is calculated by the poles of the system.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

y Output response array. Has as many rows as time samples (length of t) and as many columns as outputs.

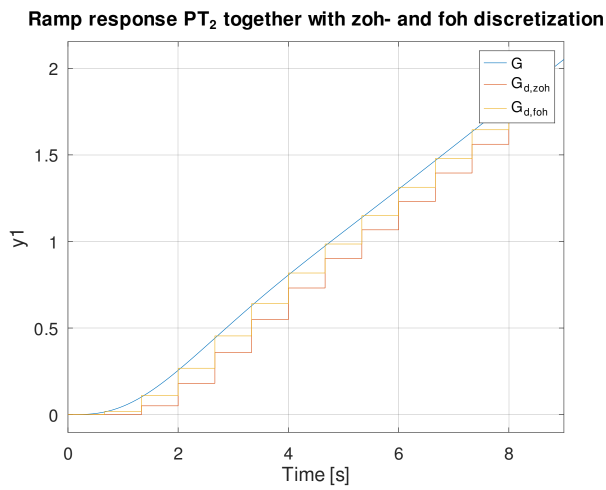
t Time row vector.

x State trajectories array. Has `length (t)` rows and as many columns as states.

See also: impulse, initial, lsim, step.

Example: 1

```
clf;
s = tf ('s');
G = 1/(2*s^2+3*s+4);
T = 2/3;
Gdzoh = c2d (G, T);
Gdfoh = c2d (G, T, 'foh');
ramp (G, Gdzoh, ';G_{d,zoh};', Gdfoh, ';G_{d,foh};');
title ("Ramp response PT_2 together with zoh- and foh discretization");
```



2.7.8 step

step (*sys*) [Function File]
step (*sys1*, *sys2*, ..., *sysN*) [Function File]
step (*sys1*, *'style1'*, ..., *sysN*, *'styleN'*) [Function File]
step (*sys1*, ..., *t*) [Function File]
step (*sys1*, ..., *tfinal*) [Function File]
step (*sys1*, ..., *tfinal*, *dt*) [Function File]
[*y*, *t*, *x*] = step (*sys*) [Function File]
[*y*, *t*, *x*] = step (*sys*, *t*) [Function File]
[*y*, *t*, *x*] = step (*sys*, *tfinal*) [Function File]

`[y, t, x] = step (sys, tfinal, dt)` [Function File]

Step response of LTI system. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI model.

t Time vector. Should be evenly spaced. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

tfinal Optional simulation horizon. If not specified, it is calculated by the poles of the system to reflect adequately the response transients.

dt Optional sampling time. Be sure to choose it small enough to capture transient phenomena. If not specified, it is calculated by the poles of the system.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

y Output response array. Has as many rows as time samples (length of t) and as many columns as outputs.

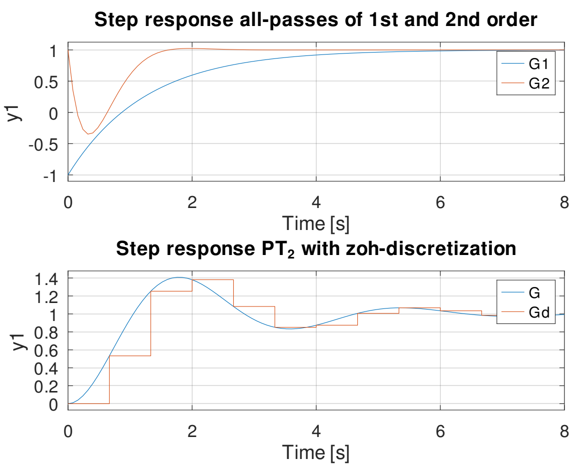
t Time row vector.

x State trajectories array. Has `length (t)` rows and as many columns as states.

See also: impulse, initial, lsim.

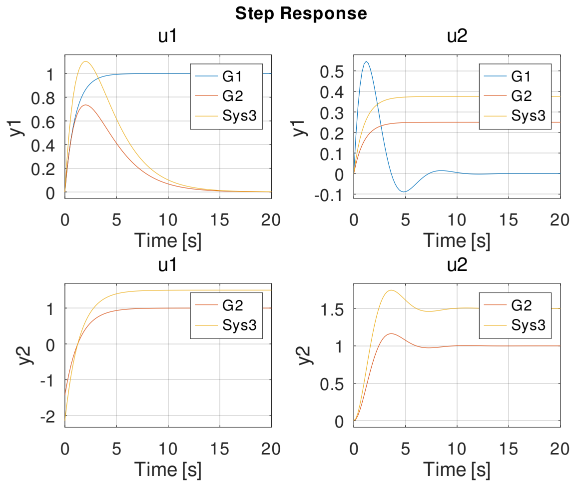
Example: 1

```
clf;
subplot (2,1,1)
s = tf('s');
G1 = (1 - 1.25*s)/(1 + 1.25*s);
G2 = (1 - .5*s + 0.1*s^2)/(1 + .5*s + 0.1*s^2);
step(G1,G2,8);
title ("Step response all-passes of 1st and 2nd order");
subplot (2,1,2)
G = 1/(0.3*s^2 + 0.3*s + 1);
Gd = c2d(G,0.6667);
step(G,Gd,8);
title ("Step response PT_2 with zoh-discretization");
```



Example: 2

```
clf;
warning ("off", "Control:convert-to-state-space");
G1 = tf({[1],[1 0]},{[1 1],[1 1 1]});
G2 = tf  ({[1 0],[0.25];[-2 1],[1]},{[1 1 0.25],[1 1];[1.4 1],[1 1 1]});
step (G1,G2,1.5*G2);
warning ("on", "Control:convert-to-state-space");
```



2.8 Frequency Domain Analysis

2.8.1 @lti/freqresp

H = `freqresp` (*sys*, *w*) [Function File]

Evaluate frequency response at given frequencies.

Inputs

sys LTI system.

w Vector of frequency values.

Outputs

H Array of frequency response. For a system with m inputs and p outputs, the array *H* has dimensions [p, m, length (w)]. The frequency response at the frequency w(k) is given by H(:,:,k).

See also: @lti/dcgain.

2.8.2 bode

`bode` (*sys*) [Function File]
`bode` (*sys1*, *sys2*, ..., *sysN*) [Function File]
`bode` (*sys1*, *'style1'*, ..., *sysN*, *'styleN'*) [Function File]
`bode` (*sys1*, *sys2*, ..., *sysN*, *w*) [Function File]
[*mag*, *pha*, *w*] = `bode` (*sys*) [Function File]
[*mag*, *pha*, *w*] = `bode` (*sys*, *w*) [Function File]

Bode diagram of frequency response. If no output arguments are given, the response is printed on the screen. In the latter case, the chidlren of the figure handle are the handles for (1) the phase plot, (2) the legend, and (3) the magnitude plot. The magnitude plot is the active handle after the bode plot is finished.

Inputs

sys LTI system. Must be a single-input and single-output (SISO) system.

w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where *wmin* and *wmax* denote minimum and maximum frequencies in rad/s.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

mag Vector of magnitude. Has length of frequency vector *w*.

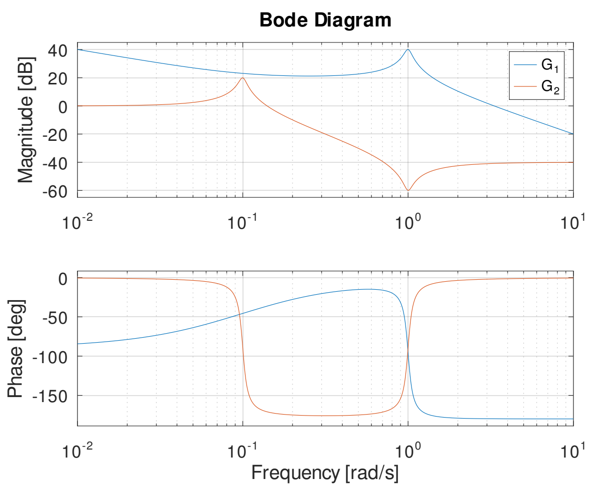
pha Vector of phase. Has length of frequency vector *w*.

w Vector of frequency values used.

See also: nichols, nyquist, sigma.

Example: 1

```
G_1 = tf ([10 1],[1 0.1 1 0]);
G_2 = tf ([1 0.1 1],[100 1 1]);
bode(G_1,G_2);
```



2.8.3 bodemag

`bodemag (`*sys*`)` [Function File]
`bodemag (`*sys1*`, `*sys2*`, ..., `*sysN*`)` [Function File]
`bodemag (`*sys1*`, `*sys2*`, ..., `*sysN*`, `*w*`)` [Function File]
`bodemag (`*sys1*`, `*'style1'*`, ..., `*sysN*`, `*'styleN'*`)` [Function File]
`[`*mag*`, `*w*`] = bodemag (`*sys*`)` [Function File]
`[`*mag*`, `*w*`] = bodemag (`*sys*`, `*w*`)` [Function File]

Bode magnitude diagram of frequency response. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI system. Must be a single-input and single-output (SISO) system.

w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where *wmin* and *wmax* denote minimum and maximum frequencies in rad/s.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

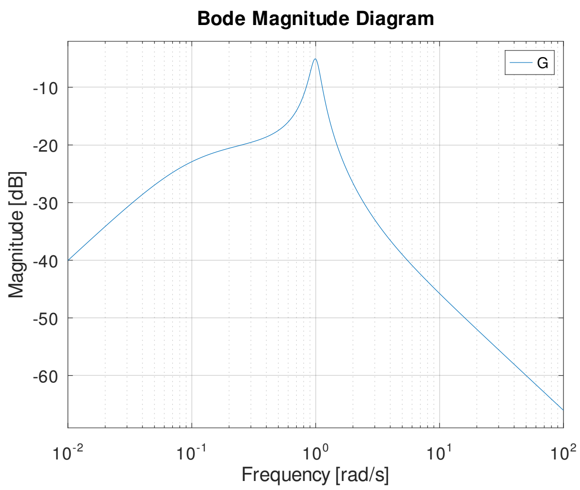
mag Vector of magnitude. Has length of frequency vector *w*.

w Vector of frequency values used.

See also: bode, nichols, nyquist, sigma.

Example: 1

```
s = tf('s');
G = s*(1 + 0.5*s)/(s^2 + 0.2*s + 1)/(1+10*s);
bodemag(G);
```



2.8.4 margin

`[gamma, phi, w_gamma, w_phi] = margin (sys)` [Function File]
`[gamma, phi, w_gamma, w_phi] = margin (sys, tol)` [Function File]

Gain and phase margin of a system. If no output arguments are given, both gain and phase margin are plotted on a bode diagram. Otherwise, the margins and their corresponding frequencies are computed and returned. A more robust criterion to assess the stability of a feedback system is the sensitivity Ms computed by function `sensitivity`.

Inputs

sys LTI model. Must be a single-input and single-output (SISO) system.

tol Imaginary parts below *tol* are assumed to be zero. If not specified, default value `sqrt (eps)` is taken.

Outputs

gamma Gain margin (as gain, not dBs).

phi Phase margin (in degrees).

w_gamma Frequency for the gain margin (in rad/s).

w_phi Frequency for the phase margin (in rad/s).

Algorithm

Uses function `roots` to calculate the frequencies *w_gamma*, *w_phi* from special polynomials created from the transfer function of *sys* as listed below in section «Equations».

Equations

```
CONTINUOUS-TIME SYSTEMS
Gain Margin
        _                _
L(jw) = L(jw)      BTW: L(jw) = L(-jw) = conj (L(jw))

num(jw)   num(-jw)
------- = --------
den(jw)   den(-jw)

num(jw) den(-jw) = num(-jw) den(jw)

imag (num(jw) den(-jw)) = 0
imag (num(-jw) den(jw)) = 0

Phase Margin
          |num(jw)|
|L(jw)| = |-------| = 1
          |den(jw)|
  _     2      2
z z = Re z + Im z

num(jw)   num(-jw)
------- * -------- = 1
den(jw)   den(-jw)

num(jw) num(-jw) - den(jw) den(-jw) = 0

real (num(jw) num(-jw) - den(jw) den(-jw)) = 0

DISCRETE-TIME SYSTEMS
Gain Margin
                              jwT         log z
L(z) = L(1/z)      BTW: z = e    --> w = -----
                                          j T
num(z)   num(1/z)
------ = --------
den(z)   den(1/z)

num(z) den(1/z) - num(1/z) den(z) = 0

Phase Margin
         |num(z)|
|L(z)| = |------| = 1
         |den(z)|
```

```
L(z) L(1/z) = 1

num(z)   num(1/z)
------ * -------- = 1
den(z)   den(1/z)

num(z) num(1/z) - den(z) den(1/z) = 0
PS: How to get L(1/z)
          4       3       2
p(z) = a z  +  b z  +  c z  +  d z  +  e

            -4      -3      -2      -1
p(1/z) = a z  +  b z  +  c z  +  d z  +  e

          -4                    2       3       4
      = z   ( a  +  b z  +  c z  +  d z  +  e z  )

              4       3       2                    4
      = ( e z  +  d z  +  c z  +  b z  +  a ) / ( z  )
```

See also: sensitivity, roots.

2.8.5 nichols

nichols (*sys*) [Function File]
nichols (*sys1*, *sys2*, ..., *sysN*) [Function File]
nichols (*sys1*, *sys2*, ..., *sysN*, *w*) [Function File]
nichols (*sys1*, *'style1'*, ..., *sysN*, *'styleN'*) [Function File]
[*mag*, *pha*, *w*] = nichols (*sys*) [Function File]
[*mag*, *pha*, *w*] = nichols (*sys*, *w*) [Function File]

Nichols chart of frequency response, plots gain over phase. If no output arguments are given, the response is printed on the screen together with a grid of contours of constant close-loop magnitude and phase.

Inputs

sys LTI system. Must be a single-input and single-output (SISO) system.

w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where *wmin* and *wmax* denote minimum and maximum frequencies in rad/s.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

mag Vector of magnitude. Has length of frequency vector *w*.

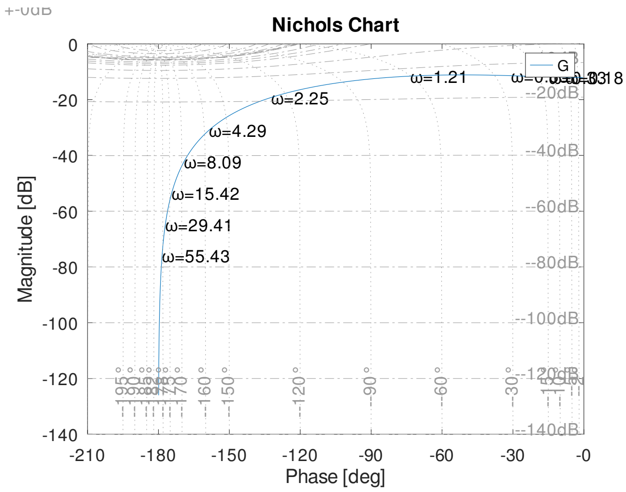
pha Vector of phase. Has length of frequency vector *w*.

w Vector of frequency values used.

See also: bode, nyquist, sigma.

Example: 1

```
s = tf('s');
G = 1/(2*s^2+3*s+4);
nichols(G);
```



2.8.6 nyquist

`nyquist (`*sys*`)` [Function File]
`nyquist (`*sys1*`, `*sys2*`, ..., `*sysN*`)` [Function File]
`nyquist (`*sys1*`, `*sys2*`, ..., `*sysN*`, `*w*`)` [Function File]
`nyquist (`*sys1*`, `*'style1'*`, ..., `*sysN*`, `*'styleN'*`)` [Function File]
`[`*re*`, `*im*`, `*w*`] = nyquist (`*sys*`)` [Function File]
`[`*re*`, `*im*`, `*w*`] = nyquist (`*sys*`, `*w*`)` [Function File]

Nyquist diagram of frequency response. If no output arguments are given, the response is printed on the screen.

Inputs

sys LTI system. Must be a single-input and single-output (SISO) system.

w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where *wmin* and *wmax* denote minimum and maximum frequencies in rad/s.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

re Vector of real parts. Has length of frequency vector *w*.

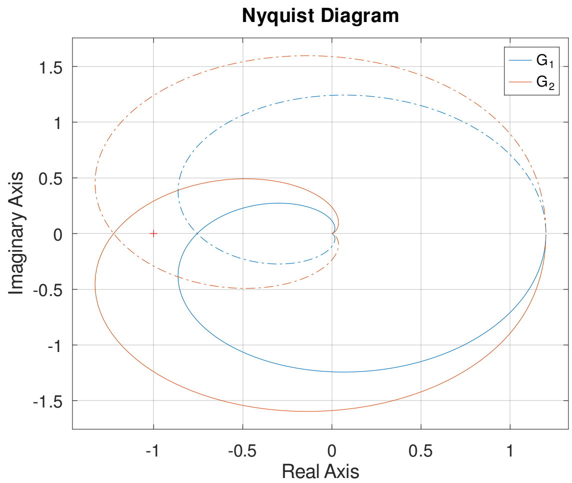
im Vector of imaginary parts. Has length of frequency vector *w*.

w Vector of frequency values used.

See also: bode, nichols, sigma.

Example: 1

```
s = tf('s');
G_1 = 1.2/(s^2/9+0.42*s+1)^2;
G_2 = 1.2/(s^2/9+0.33*s+1)^2;
nyquist(G_1,G_2);
```



2.8.7 sensitivity

`[Ms, ws] = sensitivity (L)` [Function File]
`[Ms, ws] = sensitivity (P, C)` [Function File]
`[Ms, ws] = sensitivity (P, C1, C2, ...)` [Function File]

Return sensitivity margin *Ms*. The quantity *Ms* is simply the inverse of the shortest distance from the Nyquist curve to the critical point -1. Reasonable values of *Ms* are in the range from 1.3 to 2.

$$M_s = ||S(j\omega)||_\infty$$

If no output arguments are given, the critical distance 1/Ms is plotted on a Nyquist diagram. In contrast to gain and phase margin as computed by function `margin`, the sensitivity *Ms* is a more robust criterion to assess the stability of a feedback system.

Inputs

L Open loop transfer function. *L* can be any type of LTI system, but it must be square.

P Plant model. Any type of LTI system.

C Controller model. Any type of LTI system.

C1, C2, . . .
If several controllers are specified, function `sensitivity` computes the sensitivity *Ms* for each of them in combination with plant *P*.

Outputs

Ms Sensitivity margin *Ms* as defined in [1]. Scalar value. If several controllers are specified, *Ms* becomes a row vector with as many entries as controllers.

ws The frequency [rad/s] corresponding to the sensitivity peak. Scalar value. If several controllers are specified, *ws* becomes a row vector with as many entries as controllers.

Algorithm
Uses SLICOT AB13DD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

References

1. Aström, K. and Hägglund, T. (1995) PID Controllers: Theory, Design and Tuning, Second Edition. Instrument Society of America.

2.8.8 sgrid

`sgrid` [Function File]
`sgrid on` [Function File]
`sgrid off` [Function File]
`sgrid (z, w)` [Function File]
`sgrid (hax, ...)` [Function File]

Display an grid in the complex s-plane.

Control the display of s-plane grid with :

- zeta lines corresponding to damping ratios and
- omega circles corresponding to undamped natural frequencies

The function state input may be either `"on"` or `"off"` for creating or removing the grid. If omitted, a new grid is created when it does not exist or the visibility of the current grid is toggled.

The sgrid will automatically plot the grid lines at nice values or at constant values specified by two arguments :

```
sgrid (Z, W)
```

where *Z* and *W* are :

- *Z* vector of constant zeta values to plot as lines
- *W* vector of constant omega values to plot as circles

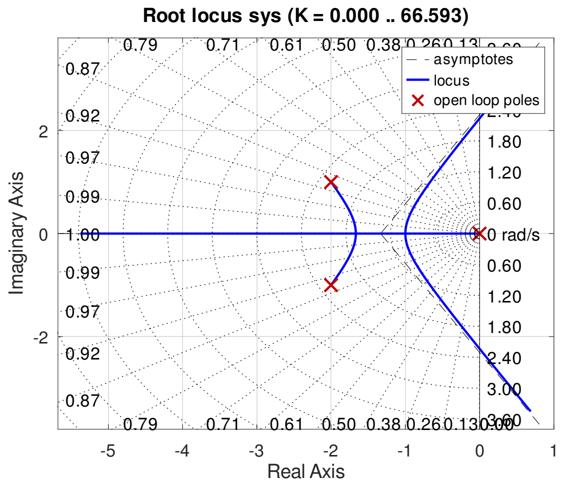
Example of usage:

```
sgrid on create the s-plane grid
sgrid off  remove the s-plane grid
sgrid toggle the s-plane grid visibility
sgrid ([0.3, 0.8, ...], [10, 75, ...])   create:
              zeta lines for 0.3, 0.8, ...
              omega circles for 10, 75, ... [rad/s]
sgrid off; sgrid  remove current s-grid and
                  draw new with default values
sgrid (hax, "on")   create the s-plane grid for the axis
                    handle hax
```

See also: grid, zgrid.

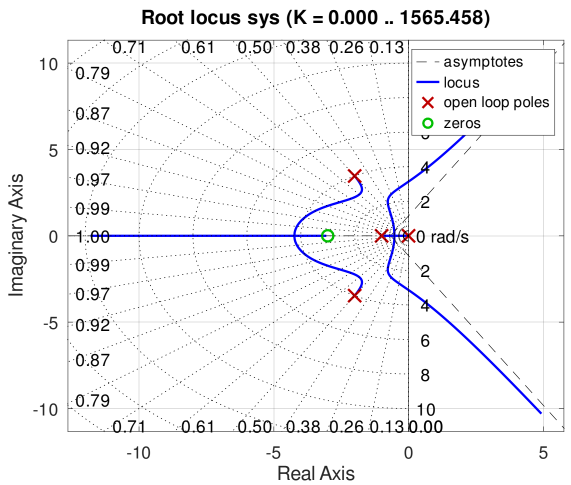
Example: 1

```
clf;
num = 1; den = [1 4 5 0];
sys = tf(num, den);
rlocus(sys);
sgrid on;
```



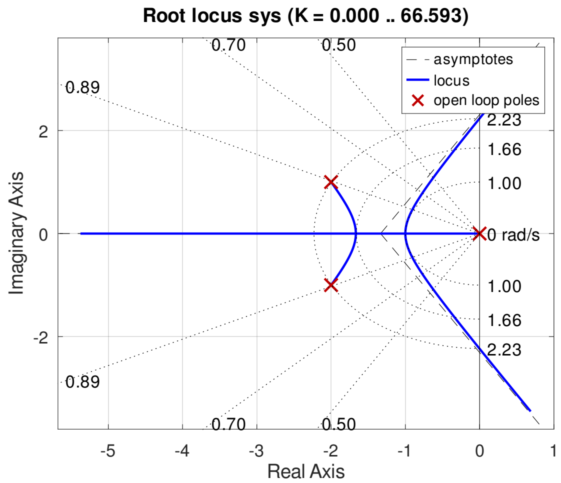
Example: 2

```
clf;
num = [1 3]; den = [1 5 20 16 0];
sys = tf(num, den);
rlocus(sys);
hfig = get(0, "currentfigure");
hax = get(hfig, "currentaxes");
sgrid(hax, "on");
```



Example: 3

```
clf;
num = 1; den = [1 4 5 0];
sys = tf(num, den);
rlocus(sys);
sgrid([0.5 0.7 0.89], [1 1.66 2.23]);
```



2.8.9 sigma

`sigma (`*sys*`)` [Function File]
`sigma (`*sys1*`, `*sys2*`, ..., `*sysN*`)` [Function File]
`sigma (`*sys1*`, `*sys2*`, ..., `*sysN*`, `*w*`)` [Function File]
`sigma (`*sys1*`, '`*style1*`', ..., `*sysN*`, '`*styleN*`')` [Function File]
`[`*sv*`, `*w*`] = sigma (`*sys*`)` [Function File]
`[`*sv*`, `*w*`] = sigma (`*sys*`, `*w*`)` [Function File]

Singular values of frequency response. If no output arguments are given, the singular value plot is printed on the screen.

Inputs

sys LTI system. Multiple inputs and/or outputs (MIMO systems) make practical sense.

w Optional vector of frequency values. If *w* is not specified, it is calculated by the zeros and poles of the system. Alternatively, the cell `{wmin, wmax}` specifies a frequency range, where *wmin* and *wmax* denote minimum and maximum frequencies in rad/s.

'style' Line style and color, e.g. 'r' for a solid red line or '-.k' for a dash-dotted black line. See `help plot` for details.

Outputs

sv Array of singular values. For a system with m inputs and p outputs, the array sv has `min (m, p)` rows and as many columns as frequency points `length (w)`. The singular values at the frequency `w(k)` are given by `sv(:,k)`.

w Vector of frequency values used.

See also: bodemag, svd.

2.8.10 zgrid

`zgrid` [Function File]
`zgrid on` [Function File]
`zgrid off` [Function File]
`zgrid (`*z*`, `*w*`)` [Function File]
`zgrid (`*hax*`, ...)` [Function File]

Display an grid in the complex z-plane.

Control the display of z-plane grid with :

- zeta lines corresponding to damping ratios and

- omega lines corresponding to undamped natural frequencies

The function state input may be either `"on"` or `"off"` for creating or removing the grid. If omitted, a new grid is created when it does not exist or the visibility of the current grid is toggled.

The zgrid will automatically plot the grid lines at nice values or at constant values specified by two arguments :

```
zgrid (Z, W)
```

where *Z* and *W* are :

- *Z* vector of constant zeta values to plot as lines
- *W* vector of constant omega values to plot as circles

Example of usage:

```
zgrid on    create the z-plane grid
zgrid off   remove the z-plane grid
zgrid    toggle the z-plane grid visibility
zgrid ([0.3, 0.8, ...], [0.25*pid, 0.5*pi, ...])   create:
```

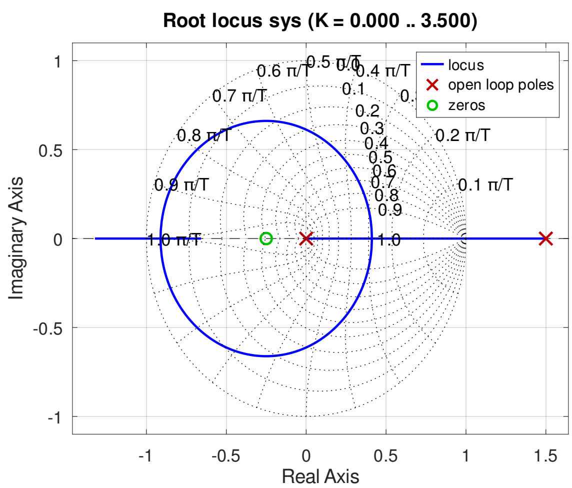
- `zeta lines for 0.3, 0.8, ...`
- `omega lines for 0.25*pi/T, 0.5*pi/T, ... [rad/s]`

```
zgrid (hax, "on")   create the z-plane grid for the axis
                    handle hax
```

See also: grid, sgrid.

Example: 1

```
clf;
num = [1 0.25]; den = [1 -1.5 0];
sys = tf(num, den, 1);
rlocus(sys,0.01,0,3.5);
ylim([-1.1,1.1]);
zgrid on;
```



2.9 PID Control

2.9.1 pid

C = pid (*Kp*) [Function File]
C = pid (*Kp*, *Ki*) [Function File]
C = pid (*Kp*, *Ki*, *Kd*) [Function File]
C = pid (*Kp*, *Ki*, *Kd*, *Tf*) [Function File]
C = pid (*Kp*, *Ki*, *Kd*, *Tf*, *Ts*) [Function File]

Return the transfer function *C* of the PID controller in parallel form with first-order roll-off. With a valid *Ts* a discrete-time system is created.

```
                 Ki       Kd s
C(s) = Kp + ---- + --------
                  s     Tf s + 1
```

2.9.2 pidstd

C = pidstd (*Kp*) [Function File]
C = pidstd (*Kp*, *Ti*) [Function File]
C = pidstd (*Kp*, *Ti*, *Td*) [Function File]
C = pidstd (*Kp*, *Ti*, *Td*, *N*) [Function File]

Return the transfer function *C* of the PID controller in standard form with first-order roll-off.

```
                    1         Td s
C(s) = Kp (1 + ---- + ----------)
                  Ti s   Td/N s + 1
```

2.10 Pole Placement

2.10.1 acker

k = acker (*A*, *b*, *p*) [Function File]

Calculates the state feedback matrix of a completely controllable SISO system using Ackermann's formula

Given the state-space system

$$\dot{x} = Ax + bu$$

and the desired eigenvalues of the closed loop in the vector *p*, the state feedback vector k is calculated in the form

$$k = (k_1 k_2 ... k_n)$$

such that the closed loop system matrix

$$A - b\,k$$

has the eigenvalues given in *p*.

See also: place.

2.10.2 place

f = place (*sys*, *p*) [Function File]
f = place (a, *b*, *p*) [Function File]
[*f*, *info*] = place (*sys*, *p*, *alpha*) [Function File]

`[f, info] = place (a, b, p, alpha)` [Function File]

Pole assignment for a given matrix pair (*A*,*B*) such that `p = eig (A-B*F)`. If parameter *alpha* is specified, poles with real parts (continuous-time) or moduli (discrete-time) below *alpha* are left untouched.

Inputs

sys Continuous- or discrete-time LTI system.

a State matrix (n-by-n) of a continuous-time system.

b Input matrix (n-by-m) of a continuous-time system.

p Desired eigenvalues of the closed-loop system state-matrix *A-B*F*. `length (p) <= rows (A)`.

alpha Specifies the maximum admissible value, either for real parts or for moduli, of the eigenvalues of *A* which will not be modified by the eigenvalue assignment algorithm. `alpha >= 0` for discrete-time systems.

Outputs

f State feedback gain matrix.

info Structure containing additional information.

info.nfp The number of fixed poles, i.e. eigenvalues of *A* having real parts less than *alpha*, or moduli less than *alpha*. These eigenvalues are not modified by `place`.

info.nap The number of assigned eigenvalues. `nap = n-nfp-nup`.

info.nup The number of uncontrollable eigenvalues detected by the eigenvalue assignment algorithm.

info.z The orthogonal matrix *z* reduces the closed-loop system state matrix `A + B*F` to upper real Schur form. Note the positive sign in `A + B*F`.

Note

```
Place is also suitable to design estimator gains:
L = place (A.', C.', p).'
L = place (sys.', p).'   # useful for discrete-time systems
```

Algorithm

Uses SLICOT SB01BD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.10.3 reg

`reg = reg (sys, k, l)` [Function File]

Form regulator from state-feedback and estimator gains

Inputs

sys State-space model of the plant

k State-feedback gain

l Estimator gain

Outputs

reg Dynamic compensator. Connect with positive feedback.

Equations See also: place.

2.10.4 rlocus

`rlocus (`*sys*`)` [Function File]
`[`*rldata*`, `*k*`] = rlocus (`*sys*`, `*increment*`, `*min_k*`, `*max_k*`)` [Function File]

Display root locus plot of the specified SISO system.

Inputs

sys LTI model. Must be a single-input and single-output (SISO) system.

increment The increment used in computing gain values.

min_k Minimum value of *k*.

max_k Maximum value of *k*.

Outputs

rldata Data points plotted: in column 1 real values, in column 2 the imaginary values.

k Gains for real axis break points.

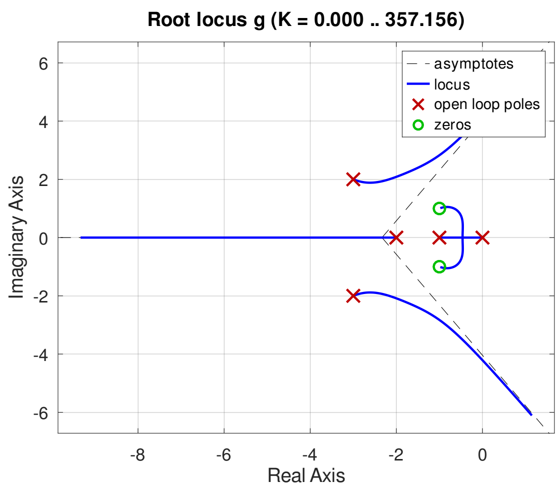
Block Diagram

```
 u    +         +---+       +-------+             y
------>(+)----->| k |----->| SISO |-------+------->
       ^ -      +---+       +-------+       |
       |                                    |
       +------------------------------------+
```

See also: rlocusx.

Example: 1

```
s = tf('s');
g = (s^2+2*s+2)/(s*(s^4+9*s^3+33*s^2+51*s+26));
rlocus(g);
```



2.10.5 rlocusx

`rlocusx (`*sys*`)` [Function File]
`rlocusx (`*sys*`, `*increment*`, `*min_k*`, `*max_k*`)` [Function File]
`rlocusx (`*sys*`, `*k_vec*`)` [Function File]

Interactive root locus plot of the specified SISO system *SYS*.

This functions directly calls rlocus() from the control package which must be installed and loaded. In contrast to rlocus(), mouse clicks on the root locus display the related gain and all other poles resulting from this gain together with damping and frequency of conjugate complex pole pairs.
All possible interaction by mouse clicks or keys are:

Left click: Gain, damping and Frequency
: Displays related gain and all resulting closed loop poles together with damping and frequency

`s`: Step response
: Simulates the step response for the gain of of the most recently selected pole locations

`i`: Impulse response
: Simulates the impulse response for the most recently selected gain

`b`: Bode plot
: Provides the open loop bode plot for the most recently selected gain

`m`: Stability margins
: Provides the open loop bode plot with stability margins for the most recently selected gain

`a`: All plots
: Provide sall four aforementioned plots

`c`: Clear
: Removes all closed loop pole markers and annotations

`d`: Delete
: Removes all open figures with simulation and bode plots

`x`: Exit
: Exits the interactive mode and re-activates the octave prompt

There are no output parameters.

Inputs

sys LTI model. Must be a single-input and single-output (SISO) system.

increment The increment used in computing gain values.

min_k Minimum value of k.

max_k Maximum value of k.

k_vec Vector of values for gain values k.

Outputs

Plots the interactive root locus to the screen.
Unlike rlocus(), this function does not have any output parameters. For output parameters please directly use rlocus().

See also: rlocus.

2.11 Optimal Control

2.11.1 augstate

augsys = augstate (*sys*) [Function File]

Append state vector x of system *sys* to output vector y.

```
.                  .
x = A x + B u      x = A x + B u
y = C x + D u  =>  y = C x + D u
                   x = I x + O u
```

2.11.2 dlqe

[*m*, *p*, *z*, *e*] = dlqe (a, *g*, *c*, *q*, *r*) [Function File]
[*m*, *p*, *z*, *e*] = dlqe (a, *g*, *c*, *q*, *r*, *s*) [Function File]
[*m*, *p*, *z*, *e*] = dlqe (a, [], *c*, *q*, *r*) [Function File]
[*m*, *p*, *z*, *e*] = dlqe (a, [], *c*, *q*, *r*, *s*) [Function File]

Kalman filter for discrete-time systems.

```
x[k] = Ax[k] + Bu[k] + Gw[k]   (State equation)
y[k] = Cx[k] + Du[k] + v[k]    (Measurement Equation)
E(w) = 0, E(v) = 0, cov(w) = Q, cov(v) = R, cov(w,v) = S
```

Inputs

a State transition matrix of discrete-time system (n-by-n).

g Process noise matrix of discrete-time system (n-by-g). If *g* is empty [], an identity matrix is assumed.

c Measurement matrix of discrete-time system (p-by-n).

q Process noise covariance matrix (g-by-g).

r Measurement noise covariance matrix (p-by-p).

s Optional cross term covariance matrix (g-by-p), s = cov(w,v). If *s* is empty [] or not specified, a zero matrix is assumed.

Outputs

m Kalman filter gain matrix (n-by-p).

p Unique stabilizing solution of the discrete-time Riccati equation (n-by-n). Symmetric matrix.

z Error covariance (n-by-n), cov(x(k|k)-x)

e Closed-loop poles (n-by-1).

Equations

```
x[k|k] = x[k|k-1] + M(y[k] - Cx[k|k-1] - Du[k])

x[k+1|k] = Ax[k|k] + Bu[k] for S=0

x[k+1|k] = Ax[k|k] + Bu[k] + G*S*(C*P*C' + R)^-1*(y[k] - C*x[k|k-1]) for non-zero S


E = eig(A - A*M*C) for S=0

E = eig(A - A*M*C - G*S*(C*P*C' + Rv)^-1*C) for non-zero S
```

See also: dare, care, dlqr, lqr, lqe.

2.11.3 dlqr

[*g*, *x*, *l*] = dlqr (*sys*, *q*, *r*) [Function File]
[*g*, *x*, *l*] = dlqr (*sys*, *q*, *r*, *s*) [Function File]
[*g*, *x*, *l*] = dlqr (a, *b*, *q*, *r*) [Function File]
[*g*, *x*, *l*] = dlqr (a, *b*, *q*, *r*, *s*) [Function File]
[*g*, *x*, *l*] = dlqr (a, *b*, *q*, *r*, [], *e*) [Function File]

`[g, x, l] = dlqr (a, b, q, r, s, e)` [Function File]

Linear-quadratic regulator for discrete-time systems.

Inputs

sys Continuous or discrete-time LTI model (p-by-m, n states).

a State transition matrix of discrete-time system (n-by-n).

b Input matrix of discrete-time system (n-by-m).

q State weighting matrix (n-by-n).

r Input weighting matrix (m-by-m).

s Optional cross term matrix (n-by-m). If *s* is not specified, a zero matrix is assumed.

e Optional descriptor matrix (n-by-n). If *e* is not specified, an identity matrix is assumed.

Outputs

g State feedback matrix (m-by-n).

x Unique stabilizing solution of the discrete-time Riccati equation (n-by-n).

l Closed-loop poles (n-by-1).

Equations

```
x[k+1] = A x[k] + B u[k],   x[0] = x0

        inf
J(x0) = SUM (x' Q x  +  u' R u  +  2 x' S u)
        k=0


L = eig (A - B*G)
```

See also: dare, care, lqr.

2.11.4 estim

`est = estim (sys, l)` [Function File]

`est = estim (sys, l, sensors, known)` [Function File]

`est = estim (sys, l, sensors, known, type)` [Function File]

Return state estimator for a given estimator gain.

Inputs

sys LTI model.

l State feedback matrix.

sensors Indices of measured output signals y from *sys*. If omitted or empty, all outputs are measured.

known Indices of known input signals u (deterministic) to *sys*. All other inputs to *sys* are assumed stochastic (w). If argument *known* is omitted or empty, no inputs u are known.

type Type of the estimator for discrete-time systems. If set to 'delayed' the current estimation is based on y(k-1), if set to 'current' the current estimation is based on the lates mesaruement y(k). If omitted, the 'delayed' version is created.

Outputs

est State-space model of estimator.

Block Diagram

```
                                 u  +-------+         ^
      +---------------------------->|       |-------> y
      |    +-------+     +       y  |  est  |         ^
u ----+--->|       |----->(+)------>|       |-------> x
           |  sys  |       ^ +      +-------+
w -------->|       |       |
           +-------+       | v
```

Remarks

The argument *type* is for discrete-time systems only. If set to 'current', the follwong prediction-correction scheme is used:

```
^             ^
x*(k+1) = A x(k) + B u(k)
   ^       ^         -1
   x(k) = x*(k) + A    L (y(k) - C x*(k) - D u(k))
```

The inverse fo the system matrix in the above equations is required for maintaining the desired observer error dynamics given by (A - LC).

The advantage of this structure is that the current measurement y(k) is used for the current estiamted state and not for the next allowing the estimator to react to system disturbances faster. L is the observer feedback matrix for the common observer structure with the matrix (A - LC) being asymptotically stable, i.e. has eigenvalues strictly within the unit circle.

See also: kalman, lqe, place.

2.11.5 kalman

`[est, g, x] = kalman (sys, Q, R)` [Function File]
`[est, g, x] = kalman (sys, Q, R, S)` [Function File]
`[est, g, x] = kalman (sys, Q, R, [], sensors, known)` [Function File]
`[est, g, x] = kalman (sys, Q, R, S, sensors, known)` [Function File]
`[est, g, x] = kalman (sys, Q, R, [], sensors, known, type)` [Function File]
`[est, g, x] = kalman (sys, Q, R, S, sensors, known, type)` [Function File]

Design Kalman estimator for LTI systems.

Inputs

sys Nominal plant model.

q Covariance of white process noise.

r Covariance of white measurement noise.

s Optional cross term covariance. Default value is 0.

sensors Indices of measured output signals y from *sys*. If omitted or empty, all outputs are measured.

known Indices of known input signals u (deterministic) to *sys*. All other inputs to *sys* are assumed stochastic. If argument *known* is omitted or empty, the first m-l inputs to *sys* are known, where m is the total number of inputs to *sys* and l is the size of the quadratic matrix *Q*.

type Type of the estimator for discrete-time systems. If set to 'delayed' the current estimation is based on y(k-1), if set to 'current' the current estimation is based on the lates mesaruement y(k). If omitted, the 'delayed' version is created.

Outputs

est State-space model of the Kalman estimator.

g Estimator gain.

x Solution of the Riccati equation.

Block Diagram

```
                                                 u  +-------+         ^
        +---------------------------->|       |-------> y
        |    +-------+     +       y  |  est  |         ^
  u ----+--->|       |----->(+)------>|       |-------> x
             |  sys  |        ^ +     +-------+
  w -------->|       |        |
             +-------+        | v

  Q = cov (w, w')     R = cov (v, v')     S = cov (w, v')
```

See also: care, dare, estim, lqr.

2.11.6 lqe

`[l, p, e] = lqe (sys, q, r)` [Function File]
`[l, p, e] = lqe (sys, q, r, s)` [Function File]
`[l, p, e] = lqe (a, g, c, q, r)` [Function File]
`[l, p, e] = lqe (a, g, c, q, r, s)` [Function File]
`[l, p, e] = lqe (a, [], c, q, r)` [Function File]
`[l, p, e] = lqe (a, [], c, q, r, s)` [Function File]

Kalman filter for continuous-time systems.

$$\dot{x} = A\,x + B\,u + G\,w$$

$$y = C\,x + D\,u + v$$

$$E(w) = 0,\ E(v) = 0,\ cov(w) = Q,\ cov(v) = R,\ cov(w, v) = S$$

Inputs

sys Continuous or discrete-time LTI model (p-by-m, n states).

a State matrix of continuous-time system (n-by-n).

g Process noise matrix of continuous-time system (n-by-g). If *g* is empty `[]`, an identity matrix is assumed.

c Measurement matrix of continuous-time system (p-by-n).

q Process noise covariance matrix (g-by-g).

r Measurement noise covariance matrix (p-by-p).

s Optional cross term covariance matrix (g-by-p), s = cov(w,v). If *s* is empty `[]` or not specified, a zero matrix is assumed.

Outputs

l Kalman filter gain matrix (n-by-p).

p Unique stabilizing solution of the continuous-time Riccati equation (n-by-n). Symmetric matrix. If *sys* is a discrete-time model, the solution of the corresponding discrete-time Riccati equation is returned.

e Closed-loop poles (n-by-1).

Equations

$$\dot{x} = A\,x + B\,u + L\,(y - C\; - D\,u)$$

$$E = \sigma(A - L\,C)$$

See also: dare, care, dlqr, lqr, dlqe.

2.11.7 lqg

reg = lqg (*sys*, *QXU*, *QWV*) [Function File]
reg = lqg (*sys*, *QXU*, *QWV*, *QI*) [Function File]

Linear-quadratic gaussian (LQG) design

Inputs

sys Continuous or discrete-time LTI model (m inputs, n states, p outputs).

QXU State and input weighting matrix (n+m-by-n+m).

QWV Process and measurement noise covariance matrix (n+p-by-n+p).

QI Optional output weighting matrix for LQG servo control with integral action (p-by-p). If *QI* is not specified, the LQG regulator is computed

Outputs

reg LQG regulator or controller as dynamic compensator. Connect with positive feedback.

Equations

$$\dot{x} = A\,x + B\,u, \quad x(0) = x_0$$

$$J(x_0) = E\left[\lim_{T\to\infty} \frac{1}{T} \int_0^T [x^T\, u^T] Q_{xu}\, [x^T\, u^T]^T + x_i^T Q_i\, x_i \; dt\right]$$

See also: lqr, kalman, lqi.

2.11.8 lqgreg

reg = lqgreg (*kest*, *k*) [Function File]

Form LQG regulator

Inputs

kest Kalman estimator

k State-feedback gain

Outputs

reg LQG regulator as dynamic compensator. Connect with positive feedback.

Equations See also: lqr, kalman, lqg.

2.11.9 lqgtrack

`reg = lqgtrack (kest, k)` [Function File]

Form LQG servo controller

Inputs

kest Kalman estimator

k State-feedback gain, including integrator states (m x n+p)

Outputs

reg LQG servo controller. Connect with positive feedback.

Equations See also: lqr, kalman, lqg, lqgreg.

2.11.10 lqi

`[g, x, l] = lqi (sys, q, r)` [Function File]
`[g, x, l] = lqr (sys, q, r, s)` [Function File]

Linear-quadratic integral control.

Inputs

sys Continuous or discrete-time LTI model (m inputs, n states, p outputs).

q State weighting matrix (n+p-by-n+p).

r Input weighting matrix (m-by-m).

s Optional cross term matrix (n+p-by-m). If *s* is not specified, a zero matrix is assumed.

Outputs

g State feedback matrix (m-by-n).

x Unique stabilizing solution of the continuous-time Riccati equation (n+p-by-n+p).

l Closed-loop poles (n-by-1).

Equations

$$\dot{x} = A\,x + B\,u, \quad x(0) = x_0$$

$$J(x_0) = \int_0^\infty z^T Q\,z + u^T R\,u + 2\,z^T S\,u\ dt$$

$$z = \begin{bmatrix} x \\ x_i \end{bmatrix}, \qquad x_i = \int_0^t r - y\ dt$$

$$L = \sigma(A - B\,G)$$

See also: care, dare, dlqr.

2.11.11 lqr

`[g, x, l] = lqr (sys, q, r)` [Function File]
`[g, x, l] = lqr (sys, q, r, s)` [Function File]
`[g, x, l] = lqr (a, b, q, r)` [Function File]
`[g, x, l] = lqr (a, b, q, r, s)` [Function File]
`[g, x, l] = lqr (a, b, q, r, [], e)` [Function File]
`[g, x, l] = lqr (a, b, q, r, s, e)` [Function File]

Linear-quadratic regulator.

Inputs

sys Continuous or discrete-time LTI model (p-by-m, n states).

a State matrix of continuous-time system (n-by-n).

b Input matrix of continuous-time system (n-by-m).

q State weighting matrix (n-by-n).

r Input weighting matrix (m-by-m).

s Optional cross term matrix (n-by-m). If *s* is not specified, a zero matrix is assumed.

e Optional descriptor matrix (n-by-n). If *e* is not specified, an identity matrix is assumed.

Outputs

g State feedback matrix (m-by-n).

x Unique stabilizing solution of the continuous-time Riccati equation (n-by-n).

l Closed-loop poles (n-by-1).

Equations

$$\dot{x} = A\,x + B\,u, \quad x(0) = x_0$$

$$J(x_0) = \int_0^\infty x^T Q\,x + u^T R\,u + 2\,x^T S\,u \;dt$$

$$L = \sigma(A - B\,G)$$

See also: care, dare, dlqr, lqry, lqi, lqg.

2.11.12 lqry

`[g, x, l] = lqry (sys, q, r, s)` [Function File]

Linear-quadratic regulator with output weighting.

Inputs

sys Continuous or discrete-time LTI model (p-by-m, n states).

q Outputs weighting matrix (p-by-p).

r Input weighting matrix (m-by-m).

s Optional cross term matrix (p-by-m). If *s* is not specified, a zero matrix is assumed.

Outputs

g State feedback matrix (m-by-n).

x Unique stabilizing solution of the continuous-time Riccati equation (n-by-n).

l Closed-loop poles (n-by-1).

Equations

$$\dot{x} = A\,x + B\,u, \quad x(0) = x_0$$

$$y = C\,x + D\,u$$

$$J(x_0) = \int_0^\infty y^T Q\,y + u^T R\,u + 2\,y^T S\,u \;dt$$

$$L = \sigma(A - B\,G)$$

See also: lqr, dare, dlqr.

2.12 Robust Control

2.12.1 augw

P = augw (*G*, *W1*, *W2*, *W3*) [Function File]

Extend plant for stacked S/KS/T problem. Subsequently, the robust control problem can be solved by h2syn or hinfsyn.

Inputs

G LTI model of plant.

W1 LTI model of performance weight. Bounds the largest singular values of sensitivity *S*. Model must be empty [], SISO or of appropriate size.

W2 LTI model to penalize large control inputs. Bounds the largest singular values of *KS*. Model must be empty [], SISO or of appropriate size.

W3 LTI model of robustness and noise sensitivity weight. Bounds the largest singular values of complementary sensitivity *T*. Model must be empty [], SISO or of appropriate size.

All inputs must be proper/realizable. Scalars, vectors and matrices are possible instead of LTI models.

Outputs

P State-space model of augmented plant.

Block Diagram

```
    | W1 | -W1*G |     z1 = W1 r  -  W1 G u
    | 0  |  W2   |     z2 =          W2   u
P = | 0  |  W3*G |     z3 =          W3 G u
    |----+-------|
    | I  |    -G |     e  =    r  -     G u

                                                      +-------+  z1
            +-------------------------------------->|  W1   |----->
            |                                         +-------+
            |                                         +-------+  z2
            |                 +---------------------->|  W2   |----->
            |                 |                       +-------+
 r   +    e |   +---------+  u |   +---------+  y       +-------+  z3
----->(+)---+-->|  K(s)   |----+-->|  G(s)   |----+---->|  W3   |----->
       ^ -      +---------+        +---------+    |     +-------+
       |                                          |
       +------------------------------------------+

               +---------+
               |         |-----> z1 (p1x1)          z1 = W1 e
 r (px1) ----->|  P(s)   |-----> z2 (p2x1)          z2 = W2 u
               |         |-----> z3 (p3x1)          z3 = W3 y
 u (mx1) ----->|         |-----> e (px1)            e = r - y
               +---------+
```

```
                +---------+
        r ----->|         |-----> z
                |  P(s)   |
        u +---->|         |-----+ e
          |     +---------+     |
          |                     |
          |     +---------+     |
          +-----|  K(s)   |<----+
                +---------+
```

References

1. Skogestad, S. and Postlethwaite I. (2005) *Multivariable Feedback Control: Analysis and Design: Second Edition*. Wiley.

See also: h2syn, hinfsyn, mixsyn.

2.12.2 fitfrd

`[sys, n] = fitfrd (dat, n)` [Function File]
`[sys, n] = fitfrd (dat, n, flag)` [Function File]

Fit frequency response data with a state-space system. If requested, the returned system is stable and minimum-phase.

Inputs

dat LTI model containing frequency response data of a SISO system.

n The desired order of the system to be fitted. `n <= length(dat.w)`.

flag The flag controls whether the returned system is stable and minimum-phase.

- *0* The system zeros and poles are not constrained. Default value.
- *1* The system zeros and poles will have negative real parts in the continuous-time case, or moduli less than 1 in the discrete-time case.

Outputs

sys State-space model of order *n*, fitted to frequency response data *dat*.

n The order of the obtained system. The value of *n* could only be modified if inputs `n > 0` and `flag = 1`.

Algorithm
Uses SLICOT SB10YD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.12.3 h2syn

`[K, N, gamma, info] = h2syn (P, nmeas, ncon)` [Function File]
`[K, N, gamma, info] = h2syn (P)` [Function File]

H-2 control synthesis for LTI plant.

Inputs

P Generalized plant. Must be a proper/realizable LTI model. If *P* is constructed with `mktito` or `augw`, arguments *nmeas* and *ncon* can be omitted.

nmeas Number of measured outputs v. The last *nmeas* outputs of *P* are connected to the inputs of controller *K*. The remaining outputs z (indices 1 to p-nmeas) are used to calculate the H-2 norm.

ncon Number of controlled inputs u. The last *ncon* inputs of *P* are connected to the outputs of controller *K*. The remaining inputs w (indices 1 to m-ncon) are excited by a harmonic test signal.

Outputs

K State-space model of the H-2 optimal controller.

N State-space model of the lower LFT of *P* and *K*.

info Structure containing additional information.

info.gamma
H-2 norm of *N*.

info.rcond Vector *rcond* contains estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller *K*. For details, see the description of the corresponding SLICOT routine.

Block Diagram

```
gamma = min||N(K)||             N = lft (P, K)
         K        2

                 +---------+
        w ----->|         |-----> z
                |  P(s)   |
        u +---->|         |-----+ v
          |      +---------+     |
          |                      |
          |      +---------+     |
          +-----|  K(s)   |<----+
                 +---------+

                 +---------+
        w ----->|  N(s)   |-----> z
                 +---------+
```

Algorithm
Uses SLICOT SB10HD and SB10ED, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: augw, lqr, dlqr, kalman.

2.12.4 hinfsyn

[*K*, *N*, *gamma*, *info*] = hinfsyn (*P*, *nmeas*, *ncon*) [Function File]
[*K*, *N*, *gamma*, *info*] = hinfsyn (*P*, *nmeas*, *ncon*, ...) [Function File]
[*K*, *N*, *gamma*, *info*] = hinfsyn (*P*, *nmeas*, *ncon*, *opt*, ...) [Function File]
[*K*, *N*, *gamma*, *info*] = hinfsyn (*P*, ...) [Function File]
[*K*, *N*, *gamma*, *info*] = hinfsyn (*P*, *opt*, ...) [Function File]

H-infinity control synthesis for LTI plant.

Inputs

P Generalized plant. Must be a proper/realizable LTI model. If *P* is constructed with `mktito` or `augw`, arguments *nmeas* and *ncon* can be omitted.

nmeas Number of measured outputs v. The last *nmeas* outputs of *P* are connected to the inputs of controller *K*. The remaining outputs z (indices 1 to p-nmeas) are used to calculate the H-infinity norm.

ncon Number of controlled inputs u. The last *ncon* inputs of *P* are connected to the outputs of controller *K*. The remaining inputs w (indices 1 to m-ncon) are excited by a harmonic test signal.

... Optional pairs of keys and values. `'key1', value1, 'key2', value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

K State-space model of the H-infinity (sub-)optimal controller.

N State-space model of the resulting closed loop system with inputs w and outputs z. It is the lower linear transformation (LFT) of *P* and *K* (see function `lft`).

info Structure containing additional information.

info.gamma
L-infinity norm of *N*.

info.rcond Vector *rcond* contains estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller *K*. For details, see the description of the corresponding SLICOT routine.

Option Keys and Values

'method' String specifying the desired kind of controller:

'optimal', 'opt', 'o'
Compute optimal controller using gamma iteration. Default selection for compatibility reasons.

'suboptimal', 'sub', 's'
Compute (sub-)optimal controller. For stability reasons, suboptimal controllers are to be preferred over optimal ones.

'gmax' The maximum value of the H-infinity norm of *N*. It is assumed that *gmax* is sufficiently large so that the controller is admissible. Default value is 1e15.

'gmin' Initial lower bound for gamma iteration. Default value is 0. *gmin* is only meaningful for optimal discrete-time controllers.

'tolgam' Tolerance used for controlling the accuracy of *gamma* and its distance to the estimated minimal possible value of *gamma*. Default value is 0.01. If *tolgam* = 0, then a default value equal to `sqrt(eps)` is used, where *eps* is the relative machine precision. For suboptimal controllers, *tolgam* is ignored.

'actol' Upper bound for the poles of the closed-loop system *N* used for determining if it is stable. *actol* >= 0 for stable systems. For suboptimal controllers, *actol* is ignored.

Block Diagram

```
gamma = min||N(K)||             N = lft (P, K)
         K         inf

               +--------+
       w ----->|        |-----> z
               |  P(s)  |
       u +---->|        |-----+ v
         |     +--------+     |
         |                    |
         |     +--------+     |
         +-----|  K(s)  |<----+
               +--------+

               +--------+
       w ----->|  N(s)  |-----> z
               +--------+
```

The signals have the following meanings:

w(t): exogenous input (reference, disturbance, ...)
u(t): control input
z(t): performance outputs (control error, control input, ...)
v(t): measured output (input of the controller)

The transfer matrix P(s) can be partitioned corresponding to the input and output signals:

```
Z(s) = P11(s) W(s) + P12(s) U(s)
V(s) = P21(s) W(s) + P22(s) U(s)

       | P11(s) : P12(s) |
P(s) = |------- :--------|
       | P21(s) : P22(s) |
                                              -1
N(S) = P11(s) + P12(s)K(s)(I - P22(s)K(s))  P21(s)
```

The state-space representation of P is given by

```
   .
  x(t) =  A x(t) +  B1 w(t) +  B2 u(t)
  z(t) = C1 x(t) + D11 w(t) + D12 u(t)
  v(t) = C2 x(t) + D21 w(t) + D22 u(t)

                          | C1 |      | D11 : D12 |
A, B = | B1 : B2 |, C = |----|, D = |-----:-----|
                          | C2 |      | D21 : D22 |
```

The used method (see **Algorithm** below) requires the following assumptions:

(A1) (A,B2) is stabilizable and (C2,A) is detectable
(A2) D12 is full column rank and D21 is full row rank,
(A3) The matrix below has full column rank for all w

```
| A-j*w*I  B2  |
|   C1     D12 |
```

(A4) The matrix below has full row rank for all w.

```
| A-j*w*I  B1  |
|   C2     D21 |
```

Algorithm

Uses SB10DD and SB10AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: augw, mixsyn, @lti/lft.

2.12.5 mixsyn

`[K, N, gamma, info] = mixsyn (G, W1, W2, W3, ...)` [Function File]

Solve stacked S/KS/T H-infinity problem. Mixed-sensitivity is the name given to transfer function shaping problems in which the sensitivity function

$$S = (I + GK)^{-1}$$

is shaped along with one or more other closed-loop transfer functions such as $K\ S$ or the complementary sensitivity function

$$T = I - S = (I + GK)^{-1}GK$$

in a typical one degree-of-freedom configuration, where G denotes the plant and K the (sub-)optimal controller to be found. The shaping of multivariable transfer functions is based on the idea that a satisfactory definition of gain (range of gain) for a matrix transfer function is given by the singular values σ of the transfer function. Hence the classical loop-shaping ideas of feedback design can be generalized to multivariable systems. In addition to the requirement that K stabilizes G, the closed-loop objectives are as follows [1]:

1. For *disturbance rejection* make $\overline{\sigma}(S)$ small.
2. For *noise attenuation* make $\overline{\sigma}(T)$ small.
3. For *reference tracking* make $\overline{\sigma}(T) \approx \underline{\sigma}(T) \approx 1$.
4. For *input usage (control energy) reduction* make $\overline{\sigma}(KS)$ small.
5. For *robust stability* in the presence of an additive perturbation $G_p = G + \Delta$ make $\overline{\sigma}(KS)$ small.
6. For *robust stability* in the presence of a multiplicative output perturbation $G_p = (I + \Delta)G$, make $\overline{\sigma}(T)$ small.

In order to find a robust controller for the so-called stacked $S/KS/T\ H_\infty$ problem, the user function `mixsyn` minimizes the following criterion

$$\min_K ||N(K)||_\infty, \quad N = |W_1 S;\ W_2 KS;\ W_3 T|$$

`[K, N] = mixsyn (G, W1, W2, W3)`. The user-defined weighting functions *W1*, *W2* and *W3* bound the largest singular values of the closed-loop transfer functions S (for performance), $K\ S$ (to penalize large inputs) and T (for robustness and to avoid sensitivity to noise), respectively [1]. A few points are to be considered when choosing the weights. The weigths *Wi* must all be proper and stable. Therefore if one wishes, for example, to minimize S at low frequencies by a weighting *W1* including integral action, $\frac{1}{s}$ needs to be approximated by $\frac{1}{s+\epsilon}$, where $\epsilon \ll 1$. Similarly one might be interested in weighting $K\ S$ with a non-proper weight *W2* to ensure that K is small outside the system bandwidth. The trick here is to replace a non-proper term such as

For more details, see [1], [2].

Inputs

G LTI model of plant.

W1 LTI model of performance weight. Bounds the largest singular values of sensitivity *S*. Model must be empty `[]`, SISO or of appropriate size.

W2 LTI model to penalize large control inputs. Bounds the largest singular values of *KS*. Model must be empty `[]`, SISO or of appropriate size.

W3 LTI model of robustness and noise sensitivity weight. Bounds the largest singular values of complementary sensitivity *T*. Model must be empty `[]`, SISO or of appropriate size.

... Optional arguments of `hinfsyn`. Type `help hinfsyn` for more information.

All inputs must be proper/realizable. Scalars, vectors and matrices are possible instead of LTI models.

Outputs

K State-space model of the H-infinity (sub-)optimal controller.

N State-space model of the lower LFT of *P* and *K*.

info Structure containing additional information.

info.gamma
L-infinity norm of *N*.

info.rcond Vector *rcond* contains estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller *K*. For details, see the description of the corresponding SLICOT routine.

Block Diagram

```
                                    | W1 S   |
gamma = min||N(K)||             N = | W2 K S | = lft (P, K)
         K         inf              | W3 T   |

                                              +-------+ z1
           +--------------------------------->|  W1   |---->
           |                                  +-------+
           |                                  +-------+ z2
           |               +----------------->|  W2   |---->
           |               |                  +-------+
 r   +    e |  +---------+ u |  +---------+ y    +-------+ z3
 --->(+)---+->|  K(s)   |---+->|  G(s)   |---+->|  W3   |---->
      ^ -     +---------+      +---------+   |  +-------+
      |                                      |
      +--------------------------------------+

               +---------+
               |         |-----> z1 (p1x1)          z1 = W1 e
 r (px1) ----->|  P(s)   |-----> z2 (p2x1)          z2 = W2 u
               |         |-----> z3 (p3x1)          z3 = W3 y
 u (mx1) ----->|         |-----> e (px1)            e = r - y
               +---------+

               +---------+
       r ----->|         |-----> z
               |  P(s)   |
       u +---->|         |-----+ e
         |     +---------+     |
         |                     |
         |     +---------+     |
         +-----|  K(s)   |<----+
               +---------+

               +---------+
       r ----->|  N(s)   |-----> z
               +---------+

Extended Plant:  P = augw (G, W1, W2, W3)
Controller:      K = mixsyn (G, W1, W2, W3)
Entire System:   N = lft (P, K)
Open Loop:       L = G * K
Closed Loop:     T = feedback (L)
```

Algorithm

Relies on functions augw and hinfsyn, which use SLICOT SB10FD, SB10DD and SB10AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

References

1. Skogestad, S. and Postlethwaite I. (2005) *Multivariable Feedback Control: Analysis and*

Design: Second Edition. Wiley, Chichester, England.

2. Meinsma, G. (1995) *Unstable and nonproper weights in H-infinity control* Automatica, Vol. 31, No. 11, pp. 1655-1658

See also: hinfsyn, augw.

2.12.6 mktito

P = `mktito` (*P*, *nmeas*, *ncon*) [Function File]

Partition LTI plant *P* for robust controller synthesis. If a plant is partitioned this way, one can omit the inputs *nmeas* and *ncon* when calling the functions `hinfsyn` and `h2syn`.

Inputs

P Generalized plant.

nmeas Number of measured outputs v. The last *nmeas* outputs of *P* are connected to the inputs of controller *K*. The remaining outputs z (indices 1 to p-nmeas) are used to calculate the H-2/H-infinity norm.

ncon Number of controlled inputs u. The last *ncon* inputs of *P* are connected to the outputs of controller *K*. The remaining inputs w (indices 1 to m-ncon) are excited by a harmonic test signal.

Outputs

P Partitioned plant. The input/output groups and names are overwritten with designations according to [1].

Block Diagram

```
min||N(K)||             N = lft (P, K)
 K         norm

               +---------+
  w = u1 ----->|         |-----> z = y1
               |  P(s)   |
  u = u2 +---->|         |-----+ y = y2
         |     +---------+     |
         |                     |
         |     +---------+     |
         +-----|  K(s)   |<----+
               +---------+

               +---------+
  w = u1 ----->|  N(s)   |-----> z = y1
               +---------+
```

Reference

1. Skogestad, S. and Postlethwaite, I. (2005) *Multivariable Feedback Control: Analysis and Design: Second Edition*. Wiley, Chichester, England.

2.12.7 ncfsyn

`[K, N, gamma, info] = ncfsyn (G, W1, W2, factor)` [Function File]

Loop shaping H-infinity synthesis. Compute positive feedback controller using the McFarlane/Glover loop shaping design procedure [1]. Using a precompensator *W1* and/or a postcompensator *W2*, the singular values of the nominal plant *G* are shaped to give a desired open-loop shape. The nominal plant *G* and shaping functions *W1*, *W2* are combined to form the shaped plant, *Gs* where `Gs = W2 G W1`. We assume that *W1* and *W2* are such that *Gs* contains no hidden modes. It is relatively easy to approximate the closed-loop requirements by the following open-loop objectives [2]:

1. For *disturbance rejection* make $\underline{\sigma}(W_2 G W_1)$ large; valid for frequencies at which $\underline{\sigma}(G_S) \gg 1$.
2. For *noise attenuation* make $\overline{\sigma}(W_2 G W_1)$ small; valid for frequencies at which $\overline{\sigma}(G_S) \ll 1$.
3. For *reference tracking* make $\underline{\sigma}(W_2 G W_1)$ large; valid for frequencies at which $\underline{\sigma}(G_S) \gg 1$.
4. For *robust stability* to a multiplicative output perturbation $G_p = (I + \Delta)G$ make $\overline{\sigma}(W_2 G W_1)$ small; valid for frequencies at which $\overline{\sigma}(G_S) \ll 1$.

Then a stabilizing controller *Ks* is synthesized for shaped plant *Gs*. The final positive feedback controller *K* is then constructed by combining the H_∞ controller *Ks* with the shaping functions *W1* and *W2* such that `K = W1 Ks W2`. In [1] is stated further that the given robust stabilization objective can be interpreted as a H_∞ problem formulation of minimizing the H_∞ norm of the frequency weighted gain from disturbances on the plant input and output to the controller input and output as follows:

$$\min_K \, ||N(K)||_\infty,$$

$$N = |W_1^{-1}; W_2 G| \ (I - KG)^{-1} \ |W_1, \ GW_2^{-1}|$$

`[K, N] = ncfsyn (G, W1, W2, f)` The function `ncfsyn` - the somewhat cryptic name stands for *normalized coprime factorization synthesis* - allows the specification of an additional argument, factor *f*. Default value `f = 1` implies that an optimal controller is required, whereas `f > 1` implies that a suboptimal controller is required, achieving a performance that is *f* times less than optimal.

Inputs

G LTI model of plant.

W1 LTI model of precompensator. Model must be SISO or of appropriate size. An identity matrix is taken if *W1* is not specified or if an empty model `[]` is passed.

W2 LTI model of postcompensator. Model must be SISO or of appropriate size. An identity matrix is taken if *W2* is not specified or if an empty model `[]` is passed.

factor `factor = 1` implies that an optimal controller is required. `factor > 1` implies that a suboptimal controller is required, achieving a performance that is *factor* times less than optimal. Default value is 1.

Outputs

K State-space model of the H-infinity loop-shaping controller. Note that *K* is a *positive* feedback controller.

N State-space model of the closed loop depicted below.

info Structure containing additional information.

info.gamma
: L-infinity norm of *N*. `gamma = norm (N, inf)`.

info.emax
: Nugap robustness. `emax = inv (gamma)`.

info.Gs
: Shaped plant. `Gs = W2 * G * W1`.

info.Ks
: Controller for shaped plant. `Ks = ncfsyn (Gs)`.

info.rcond
: Estimates of the reciprocal condition numbers of the Riccati equations and a few other things. For details, see the description of the corresponding SLICOT routine.

Block Diagram of N

```
            ^ z1               ^ z2
            |                  |
 w1  +      |   +---------+    |           +---------+
----->(+)---+-->|   Ks    |----+--->(+)---->|   Gs    |----+
       ^ +      +---------+          ^      +---------+    |
       |                        w2   |                     |
       |                                                   |
       +---------------------------------------------------+
```

Algorithm

Uses SB10KD and SB10ZD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

References

1. D. McFarlane and K. Glover, *A Loop Shaping Design Procedure Using H-infinity Synthesis*, IEEE Transactions on Automatic Control, Vol. 37, No. 6, June 1992.

2. S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design: Second Edition*. Wiley, Chichester, England, 2005.

2.13 Matrix Equation Solvers

2.13.1 care

`[x, l, g] = care (a, b, q, r)` [Function File]
`[x, l, g] = care (a, b, q, r, s)` [Function File]
`[x, l, g] = care (a, b, q, r, [], e)` [Function File]
`[x, l, g] = care (a, b, q, r, s, e)` [Function File]

Solve continuous-time algebraic Riccati equation (ARE).

Inputs

a Real matrix (n-by-n).

b Real matrix (n-by-m).

q Real matrix (n-by-n).

r Real matrix (m-by-m).

s Optional real matrix (n-by-m). If *s* is not specified, a zero matrix is assumed.

e Optional descriptor matrix (n-by-n). If *e* is not specified, an identity matrix is assumed.

Outputs

x Unique stabilizing solution of the continuous-time Riccati equation (n-by-n).

l Closed-loop poles (n-by-1).

g Corresponding gain matrix (m-by-n).

Equations

```
                     -1
A'X + XA - XB R  B'X + Q = 0

                           -1
A'X + XA - (XB + S) R  (B'X + S') + Q = 0

     -1
G = R  B'X

     -1
G = R  (B'X + S')

L = eig (A - B*G)
                          -1
A'XE + E'XA - E'XB R   B'XE + Q = 0

                                -1
A'XE + E'XA - (E'XB + S) R   (B'XE + S') + Q = 0

     -1
G = R  B'XE

     -1
G = R  (B'XE + S)

L = eig (A - B*G, E)
```

Algorithm
Uses SLICOT SB02OD and SG02AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: dare, lqr, dlqr, kalman.

2.13.2 dare

[*x*, *l*, *g*] = dare (*a*, *b*, *q*, *r*) [Function File]
[*x*, *l*, *g*] = dare (*a*, *b*, *q*, *r*, *s*) [Function File]
[*x*, *l*, *g*] = dare (*a*, *b*, *q*, *r*, [], *e*) [Function File]
[*x*, *l*, *g*] = dare (*a*, *b*, *q*, *r*, *s*, *e*) [Function File]

Solve discrete-time algebraic Riccati equation (ARE).

Inputs

a Real matrix (n-by-n).

b Real matrix (n-by-m).

q Real matrix (n-by-n).

r Real matrix (m-by-m).

s Optional real matrix (n-by-m). If *s* is not specified, a zero matrix is assumed.

e Optional descriptor matrix (n-by-n). If *e* is not specified, an identity matrix is assumed.

Outputs

x Unique stabilizing solution of the discrete-time Riccati equation (n-by-n).

l Closed-loop poles (n-by-1).

g Corresponding gain matrix (m-by-n).

Equations

```
                           -1
A'XA - X - A'XB (B'XB + R)   B'XA + Q = 0

                                 -1
A'XA - X - (A'XB + S) (B'XB + R)   (B'XA + S') + Q = 0

               -1
G = (B'XB + R)   B'XA

               -1
G = (B'XB + R)   (B'XA + S')

L = eig (A - B*G)
                              -1
A'XA - E'XE - A'XB (B'XB + R)   B'XA + Q = 0

                                    -1
A'XA - E'XE - (A'XB + S) (B'XB + R)   (B'XA + S') + Q = 0

               -1
G = (B'XB + R)   B'XA

               -1
G = (B'XB + R)   (B'XA + S')

L = eig (A - B*G, E)
```

Algorithm
Uses SLICOT SB02OD and SG02AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: care, lqr, dlqr, kalman.

2.13.3 dlyap

`x = dlyap (a, b)` [Function File]
`x = dlyap (a, b, c)` [Function File]
`x = dlyap (a, b, [], e)` [Function File]

Solve discrete-time Lyapunov or Sylvester equations.

Equations

```
AXA' - X + B = 0      (Lyapunov Equation)

AXB - X + C = 0       (Sylvester Equation)

AXA' - EXE' + B = 0   (Generalized Lyapunov Equation)
```

Algorithm
Uses SB04QD and SG03AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: dlyapchol, lyap, lyapchol.

2.13.4 dlyapchol

u = dlyapchol (*a*, *b*) [Function File]
u = dlyapchol (*a*, *b*, *e*) [Function File]

Compute Cholesky factor of discrete-time Lyapunov equations.

Equations

```
A U' U A'  -  U' U  +  B B'  =  0           (Lyapunov Equation)

A U' U A'  -  E U' U E'  +  B B'  =  0      (Generalized Lyapunov Equation)
```

Algorithm
Uses SLICOT SB03OD and SG03BD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: dlyap, lyap, lyapchol.

2.13.5 lyap

x = lyap (*a*, *b*) [Function File]
x = lyap (*a*, *b*, *c*) [Function File]
x = lyap (*a*, *b*, [], *e*) [Function File]

Solve continuous-time Lyapunov or Sylvester equations.

Equations

```
AX + XA' + B = 0      (Lyapunov Equation)

AX + XB + C = 0       (Sylvester Equation)

AXE' + EXA' + B = 0   (Generalized Lyapunov Equation)
```

Algorithm
Uses SLICOT SB03MD SB04MD and SG03AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: lyapchol, dlyap, dlyapchol.

2.13.6 lyapchol

u = lyapchol (*a*, *b*) [Function File]
u = lyapchol (*a*, *b*, *e*) [Function File]

Compute Cholesky factor of continuous-time Lyapunov equations.

Equations

```
A U' U  +  U' U A'  +  B B'  =  0           (Lyapunov Equation)

A U' U E'  +  E U' U A'  +  B B'  =  0      (Generalized Lyapunov Equation)
```

Algorithm
Uses SLICOT SB03OD and SG03BD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

See also: lyap, dlyap, dlyapchol.

2.14 Model Reduction

2.14.1 bstmodred

`[Gr, info] = bstmodred (G, ...)` [Function File]
`[Gr, info] = bstmodred (G, nr, ...)` [Function File]
`[Gr, info] = bstmodred (G, opt, ...)` [Function File]
`[Gr, info] = bstmodred (G, nr, opt, ...)` [Function File]

Model order reduction by Balanced Stochastic Truncation (BST) method. The aim of model reduction is to find an LTI system *Gr* of order *nr* (nr < n) such that the input-output behaviour of *Gr* approximates the one from original system *G*.

BST is a relative error method which tries to minimize

$$||G^{-1}(G-G_r)||_\infty = \min$$

Inputs

G LTI model to be reduced.

nr The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is chosen automatically according to the description of key *'order'*.

... Optional pairs of keys and values. `"key1", value1, "key2", value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

Gr Reduced order state-space model.

info Struct containing additional information.

info.n The order of the original system *G*.

info.ns The order of the *alpha*-stable subsystem of the original system *G*.

info.hsv The Hankel singular values of the phase system corresponding to the *alpha*-stable part of the original system *G*. The *ns* Hankel singular values are ordered decreasingly.

info.nu The order of the *alpha*-unstable subsystem of both the original system *G* and the reduced-order system *Gr*.

info.nr The order of the obtained reduced order system *Gr*.

Option Keys and Values

'order', 'nr'
The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is the sum of NU and the number of Hankel singular values greater than `MAX(TOL1,NS*EPS)`; *nr* can be further reduced to ensure that `HSV(NR-NU) > HSV(NR+1-NU)`.

'method' Approximation method for the H-infinity norm. Valid values corresponding to this key are:

'sr-bta', 'b'
Use the square-root Balance & Truncate method.

'bfsr-bta', 'f'
Use the balancing-free square-root Balance & Truncate method. Default method.

'sr-spa', 's'
Use the square-root Singular Perturbation Approximation method.

'bfsr-spa', 'p'
Use the balancing-free square-root Singular Perturbation Approximation method.

'alpha' Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix $G.A$. For a continuous-time system, ALPHA <= 0 is the boundary value for the real parts of eigenvalues, while for a discrete-time system, 0 <= ALPHA <= 1 represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time systems and 1 for discrete-time systems.

'beta' Use `[G, beta*I]` as new system G to combine absolute and relative error methods. BETA > 0 specifies the absolute/relative error weighting parameter. A large positive value of BETA favours the minimization of the absolute approximation error, while a small value of BETA is appropriate for the minimization of the relative error. BETA = 0 means a pure relative error method and can be used only if rank(G.D) = rows(G.D) which means that the feedthrough matrice must not be rank-deficient. Default value is 0.

'tol1' If *'order'* is not specified, *tol1* contains the tolerance for determining the order of reduced system. For model reduction, the recommended value of *tol1* lies in the interval [0.00001, 0.001]. *tol1* < 1. If *tol1* <= 0 on entry, the used default value is *tol1* = NS*EPS, where NS is the number of ALPHA-stable eigenvalues of A and EPS is the machine precision. If *'order'* is specified, the value of *tol1* is ignored.

'tol2' The tolerance for determining the order of a minimal realization of the phase system (see METHOD) corresponding to the ALPHA-stable part of the given system. The recommended value is TOL2 = NS*EPS. TOL2 <= TOL1 < 1. This value is used by default if *'tol2'* is not specified or if TOL2 <= 0 on entry.

'equil', 'scale'
Boolean indicating whether equilibration (scaling) should be performed on system G prior to order reduction. Default value is true if `G.scaled == false` and false if `G.scaled == true`. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` function because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

BST is often suitable to perform model reduction in order to obtain low order design models for controller synthesis.

Approximation Properties:

- Guaranteed stability of reduced models

- Approximates simultaneously gain and phase
- Preserves non-minimum phase zeros
- Guaranteed a priori error bound

$$||G^{-1}(G - G_r)||_\infty \leq 2 \sum_{j=r+1}^{n} \frac{1 + \sigma_j}{1 - \sigma_j} - 1$$

Algorithm

Uses SLICOT AB09HD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.14.2 btamodred

`[Gr, info] = btamodred (G, ...)` [Function File]
`[Gr, info] = btamodred (G, nr, ...)` [Function File]
`[Gr, info] = btamodred (G, opt, ...)` [Function File]
`[Gr, info] = btamodred (G, nr, opt, ...)` [Function File]

Model order reduction by frequency weighted Balanced Truncation Approximation (BTA) method. The aim of model reduction is to find an LTI system *Gr* of order *nr* (nr < n) such that the input-output behaviour of *Gr* approximates the one from original system *G*.

BTA is an absolute error method which tries to minimize

$$||G - G_r||_\infty = \min$$

$$||V\ (G - G_r)\ W||_\infty = \min$$

where V and W denote output and input weightings.

Inputs

G LTI model to be reduced.

nr The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is chosen automatically according to the description of key *'order'*.

... Optional pairs of keys and values. `"key1", value1, "key2", value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

Gr Reduced order state-space model.

info Struct containing additional information.

- *info.n* The order of the original system *G*.
- *info.ns* The order of the *alpha*-stable subsystem of the original system *G*.
- *info.hsv* The Hankel singular values of the *alpha*-stable part of the original system *G*, ordered decreasingly.
- *info.nu* The order of the *alpha*-unstable subsystem of both the original system *G* and the reduced-order system *Gr*.
- *info.nr* The order of the obtained reduced order system *Gr*.

Option Keys and Values

'order', 'nr'
: The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is chosen automatically such that states with Hankel singular values *info.hsv* > *tol1* are retained.

'left', 'output'
: LTI model of the left/output frequency weighting *V*. Default value is an identity matrix.

'right', 'input'
: LTI model of the right/input frequency weighting *W*. Default value is an identity matrix.

'method'
: Approximation method for the L-infinity norm to be used as follows:

  'sr', 'b' Use the square-root Balance & Truncate method.

  'bfsr', 'f' Use the balancing-free square-root Balance & Truncate method. Default method.

'alpha'
: Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix *G.A*. For a continuous-time system, ALPHA <= 0 is the boundary value for the real parts of eigenvalues, while for a discrete-time system, 0 <= ALPHA <= 1 represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time systems and 1 for discrete-time systems.

'tol1'
: If *'order'* is not specified, *tol1* contains the tolerance for determining the order of the reduced model. For model reduction, the recommended value of *tol1* is c*info.hsv(1), where c lies in the interval [0.00001, 0.001]. Default value is info.ns*eps*info.hsv(1). If *'order'* is specified, the value of *tol1* is ignored.

'tol2'
: The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given model. TOL2 <= TOL1. If not specified, ns*eps*info.hsv(1) is chosen.

'gram-ctrb'
: Specifies the choice of frequency-weighted controllability Grammian as follows:

  'standard' Choice corresponding to a combination method [4] of the approaches of Enns [1] and Lin-Chiu [2,3]. Default method.

  'enhanced'
  Choice corresponding to the stability enhanced modified combination method of [4].

'gram-obsv'
: Specifies the choice of frequency-weighted observability Grammian as follows:

  'standard' Choice corresponding to a combination method [4] of the approaches of Enns [1] and Lin-Chiu [2,3]. Default method.

  'enhanced'
  Choice corresponding to the stability enhanced modified combination method of [4].

'alpha-ctrb'
: Combination method parameter for defining the frequency-weighted controllability Grammian. abs(alphac) <= 1. If alphac = 0, the choice of Grammian

corresponds to the method of Enns [1], while if alphac = 1, the choice of Grammian corresponds to the method of Lin and Chiu [2,3]. Default value is 0.

'alpha-obsv'
: Combination method parameter for defining the frequency-weighted observability Grammian. abs(alphao) <= 1. If alphao = 0, the choice of Grammian corresponds to the method of Enns [1], while if alphao = 1, the choice of Grammian corresponds to the method of Lin and Chiu [2,3]. Default value is 0.

'equil', 'scale'
: Boolean indicating whether equilibration (scaling) should be performed on system *G* prior to order reduction. This is done by state transformations. Default value is true if `G.scaled == false` and false if `G.scaled == true`. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` function because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Approximation Properties:

- Guaranteed stability of reduced models
- Lower guaranteed error bound
- Guaranteed a priori error bound

$$\sigma_{r+1} \leq ||(G - G_r)||_\infty \leq 2 \sum_{j=r+1}^{n} \sigma_j$$

References

1. Enns, D. *Model reduction with balanced realizations: An error bound and a frequency weighted generalization*. Proc. 23-th CDC, Las Vegas, pp. 127-132, 1984.
2. Lin, C.-A. and Chiu, T.-Y. *Model reduction via frequency-weighted balanced realization*. Control Theory and Advanced Technology, vol. 8, pp. 341-351, 1992.
3. Sreeram, V., Anderson, B.D.O and Madievski, A.G. *New results on frequency weighted balanced reduction technique*. Proc. ACC, Seattle, Washington, pp. 4004-4009, 1995.
4. Varga, A. and Anderson, B.D.O. *Square-root balancing-free methods for the frequency-weighted balancing related model reduction*. (report in preparation)

Algorithm

Uses SLICOT AB09ID, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.14.3 hnamodred

`[Gr, info] = hnamodred (G, ...)` [Function File]
`[Gr, info] = hnamodred (G, nr, ...)` [Function File]
`[Gr, info] = hnamodred (G, opt, ...)` [Function File]
`[Gr, info] = hnamodred (G, nr, opt, ...)` [Function File]

Model order reduction by frequency weighted optimal Hankel-norm (HNA) method. The aim of model reduction is to find an LTI system *Gr* of order *nr* (nr < n) such that the input-output behaviour of *Gr* approximates the one from original system *G*.

HNA is an absolute error method which tries to minimize

$$||G - G_r||_H = \min$$

$$||V\ (G - G_r)\ W||_H = \min$$

where V and W denote output and input weightings.

Inputs

G LTI model to be reduced.

nr The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is chosen automatically according to the description of key "*order*".

... Optional pairs of keys and values. `"key1", value1, "key2", value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

Gr Reduced order state-space model.

info Struct containing additional information.

info.n The order of the original system *G*.

info.ns The order of the *alpha*-stable subsystem of the original system *G*.

info.hsv The Hankel singular values corresponding to the projection `op(V)*G1*op(W)`, where G1 denotes the *alpha*-stable part of the original system *G*. The *ns* Hankel singular values are ordered decreasingly.

info.nu The order of the *alpha*-unstable subsystem of both the original system *G* and the reduced-order system *Gr*.

info.nr The order of the obtained reduced order system *Gr*.

Option Keys and Values

'order', 'nr'
The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is the sum of *info.nu* and the number of Hankel singular values greater than `max(tol1, ns*eps*info.hsv(1);`

'method' Specifies the computational approach to be used. Valid values corresponding to this key are:

'descriptor'
Use the inverse free descriptor system approach.

'standard' Use the inversion based standard approach.

'auto' Switch automatically to the inverse free descriptor approach in case of badly conditioned feedthrough matrices in V or W. Default method.

'left', 'v' LTI model of the left/output frequency weighting. The weighting must be antistable. $||V\ (G - G_r) \dots ||_H = \min$

'right', 'w' LTI model of the right/input frequency weighting. The weighting must be antistable. $|| \dots (G - G_r)\ W||_H = \min$

'left-inv', 'inv-v'
LTI model of the left/output frequency weighting. The weighting must have only antistable zeros. $||V^{-1}(G - G_r) \dots ||_H = \min$

'right-inv', 'inv-w'
LTI model of the right/input frequency weighting. The weighting must have only antistable zeros. $|| \dots (G - G_r)\, W^{-1}||_H = \min$

'left-conj', 'conj-v'
LTI model of the left/output frequency weighting. The weighting must be stable. $||conj(V)\,(G - G_r) \dots ||_H = \min$

'right-conj', 'conj-w'
LTI model of the right/input frequency weighting. The weighting must be stable. $|| \dots (G - G_r)\ conj(W)||_H = \min$

'left-conj-inv', 'conj-inv-v'
LTI model of the left/output frequency weighting. The weighting must be minimum-phase. $||conj(V^{-1})\,(G - G_r) \dots ||_H = \min$

'right-conj-inv', 'conj-inv-w'
LTI model of the right/input frequency weighting. The weighting must be minimum-phase. $|| \dots (G - G_r)\ conj(W^{-1})||_H = \min$

'alpha' Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix $G.A$. For a continuous-time system, ALPHA `<=` 0 is the boundary value for the real parts of eigenvalues, while for a discrete-time system, 0 `<=` ALPHA `<=` 1 represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time systems and 1 for discrete-time systems.

'tol1' If *'order'* is not specified, *tol1* contains the tolerance for determining the order of the reduced model. For model reduction, the recommended value of *tol1* is c*info.hsv(1), where c lies in the interval [0.00001, 0.001]. *tol1* `<` 1. If *'order'* is specified, the value of *tol1* is ignored.

'tol2' The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given model. *tol2* `<=` *tol1* `<` 1. If not specified, ns*eps*info.hsv(1) is chosen.

'equil', 'scale'
Boolean indicating whether equilibration (scaling) should be performed on system G prior to order reduction. Default value is true if `G.scaled == false` and false if `G.scaled == true`. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` function because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Approximation Properties:

- Guaranteed stability of reduced models
- Lower guaranteed error bound
- Guaranteed a priori error bound

$$\sigma_{r+1} \leq ||(G - G_r)||_\infty \leq 2 \sum_{j=r+1}^{n} \sigma_j$$

Algorithm
Uses SLICOT AB09JD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.14.4 spamodred

`[Gr, info] = spamodred (G, ...)` [Function File]
`[Gr, info] = spamodred (G, nr, ...)` [Function File]
`[Gr, info] = spamodred (G, opt, ...)` [Function File]
`[Gr, info] = spamodred (G, nr, opt, ...)` [Function File]

Model order reduction by frequency weighted Singular Perturbation Approximation (SPA). The aim of model reduction is to find an LTI system *Gr* of order *nr* (nr < n) such that the input-output behaviour of *Gr* approximates the one from original system *G*.

SPA is an absolute error method which tries to minimize

$$||G - G_r||_\infty = \min$$

$$||V\ (G - G_r)\ W||_\infty = \min$$

where V and W denote output and input weightings.

Inputs

G LTI model to be reduced.

nr The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is chosen automatically according to the description of key *'order'*.

... Optional pairs of keys and values. `"key1", value1, "key2", value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

Gr Reduced order state-space model.

info Struct containing additional information.

- *info.n* The order of the original system *G*.
- *info.ns* The order of the *alpha*-stable subsystem of the original system *G*.
- *info.hsv* The Hankel singular values of the *alpha*-stable part of the original system *G*, ordered decreasingly.
- *info.nu* The order of the *alpha*-unstable subsystem of both the original system *G* and the reduced-order system *Gr*.
- *info.nr* The order of the obtained reduced order system *Gr*.

Option Keys and Values

'order', 'nr'
The desired order of the resulting reduced order system *Gr*. If not specified, *nr* is chosen automatically such that states with Hankel singular values *info.hsv* > *tol1* are retained.

'left', 'output'
LTI model of the left/output frequency weighting *V*. Default value is an identity matrix.

'right', 'input'
LTI model of the right/input frequency weighting *W*. Default value is an identity matrix.

'method' Approximation method for the L-infinity norm to be used as follows:

'sr', 's' Use the square-root Singular Perturbation Approximation method.

'bfsr', 'p' Use the balancing-free square-root Singular Perturbation Approximation method. Default method.

'alpha' Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix *G.A*. For a continuous-time system, ALPHA <= 0 is the boundary value for the real parts of eigenvalues, while for a discrete-time system, 0 <= ALPHA <= 1 represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time systems and 1 for discrete-time systems.

'tol1' If *'order'* is not specified, *tol1* contains the tolerance for determining the order of the reduced model. For model reduction, the recommended value of *tol1* is c*info.hsv(1), where c lies in the interval [0.00001, 0.001]. Default value is info.ns*eps*info.hsv(1). If *'order'* is specified, the value of *tol1* is ignored.

'tol2' The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given model. TOL2 <= TOL1. If not specified, ns*eps*info.hsv(1) is chosen.

'gram-ctrb'
Specifies the choice of frequency-weighted controllability Grammian as follows:

'standard' Choice corresponding to a combination method [4] of the approaches of Enns [1] and Lin-Chiu [2,3]. Default method.

'enhanced'
Choice corresponding to the stability enhanced modified combination method of [4].

'gram-obsv'
Specifies the choice of frequency-weighted observability Grammian as follows:

'standard' Choice corresponding to a combination method [4] of the approaches of Enns [1] and Lin-Chiu [2,3]. Default method.

'enhanced'
Choice corresponding to the stability enhanced modified combination method of [4].

'alpha-ctrb'
Combination method parameter for defining the frequency-weighted controllability Grammian. abs(alphac) <= 1. If alphac = 0, the choice of Grammian corresponds to the method of Enns [1], while if alphac = 1, the choice of Grammian corresponds to the method of Lin and Chiu [2,3]. Default value is 0.

'alpha-obsv'
Combination method parameter for defining the frequency-weighted observability Grammian. abs(alphao) <= 1. If alphao = 0, the choice of Grammian corresponds to the method of Enns [1], while if alphao = 1, the choice of Grammian corresponds to the method of Lin and Chiu [2,3]. Default value is 0.

'equil', 'scale'
: Boolean indicating whether equilibration (scaling) should be performed on system *G* prior to order reduction. Default value is true if `G.scaled == false` and false if `G.scaled == true`. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` function because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

References

1. Enns, D. *Model reduction with balanced realizations: An error bound and a frequency weighted generalization*. Proc. 23-th CDC, Las Vegas, pp. 127-132, 1984.
2. Lin, C.-A. and Chiu, T.-Y. *Model reduction via frequency-weighted balanced realization*. Control Theory and Advanced Technology, vol. 8, pp. 341-351, 1992.
3. Sreeram, V., Anderson, B.D.O and Madievski, A.G. *New results on frequency weighted balanced reduction technique*. Proc. ACC, Seattle, Washington, pp. 4004-4009, 1995.
4. Varga, A. and Anderson, B.D.O. *Square-root balancing-free methods for the frequency-weighted balancing related model reduction*. (report in preparation)

Algorithm
Uses SLICOT AB09ID, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.15 Controller Reduction

2.15.1 btaconred

`[Kr, info] = btaconred (G, K, ...)` [Function File]
`[Kr, info] = btaconred (G, K, ncr, ...)` [Function File]
`[Kr, info] = btaconred (G, K, opt, ...)` [Function File]
`[Kr, info] = btaconred (G, K, ncr, opt, ...)` [Function File]

Controller reduction by frequency-weighted Balanced Truncation Approximation (BTA). Given a plant *G* and a stabilizing controller *K*, determine a reduced order controller *Kr* such that the closed-loop system is stable and closed-loop performance is retained.

The algorithm tries to minimize the frequency-weighted error

$$||V\ (K - K_r)\ W||_\infty = \min$$

where V and W denote output and input weightings.

Inputs

G
: LTI model of the plant. It has m inputs, p outputs and n states.

K
: LTI model of the controller. It has p inputs, m outputs and nc states.

ncr
: The desired order of the resulting reduced order controller *Kr*. If not specified, *ncr* is chosen automatically according to the description of key *'order'*.

...
: Optional pairs of keys and values. `"key1", value1, "key2", value2`.

opt
: Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

Kr State-space model of reduced order controller.

info Struct containing additional information.

- *info.ncr* The order of the obtained reduced order controller *Kr*.
- *info.ncs* The order of the alpha-stable part of original controller *K*.
- *info.hsvc* The Hankel singular values of the alpha-stable part of *K*. The *ncs* Hankel singular values are ordered decreasingly.

Option Keys and Values

'order', 'ncr'
The desired order of the resulting reduced order controller *Kr*. If not specified, *ncr* is chosen automatically such that states with Hankel singular values *info.hsvc* > *tol1* are retained.

'method' Order reduction approach to be used as follows:

- *'sr', 'b'* Use the square-root Balance & Truncate method.
- *'bfsr', 'f'* Use the balancing-free square-root Balance & Truncate method. Default method.

'weight' Specifies the type of frequency-weighting as follows:

- *'none'* No weightings are used (V = I, W = I).
- *'left', 'output'*
  Use stability enforcing left (output) weighting

$$V = (I - GK)^{-1}G, \qquad W = I$$

- *'right', 'input'*
  Use stability enforcing right (input) weighting

$$V = I, \qquad W = (I - GK)^{-1}G$$

- *'both', 'performance'*
  Use stability and performance enforcing weightings

$$V = (I - GK)^{-1}G, \qquad W = (I - GK)^{-1}$$

  Default value.

'feedback' Specifies whether *K* is a positive or negative feedback controller:

- *'+'* Use positive feedback controller. Default value.
- *'-'* Use negative feedback controller.

'alpha' Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix *K.A*. For a continuous-time controller, ALPHA <= 0 is the boundary value for the real parts of eigenvalues, while for a discrete-time controller, 0 <= ALPHA <= 1 represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time controllers and 1 for discrete-time controllers.

'tol1' If *'order'* is not specified, *tol1* contains the tolerance for determining the order of the reduced controller. For model reduction, the recommended value of *tol1* is c*info.hsvc(1), where c lies in the interval [0.00001, 0.001]. Default value is info.ncs*eps*info.hsvc(1). If *'order'* is specified, the value of *tol1* is ignored.

'tol2' The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given controller. TOL2 <= TOL1. If not specified, ncs*eps*info.hsvc(1) is chosen.

'gram-ctrb'
Specifies the choice of frequency-weighted controllability Grammian as follows:

'standard' Choice corresponding to standard Enns' method [1]. Default method.

'enhanced'
Choice corresponding to the stability enhanced modified Enns' method of [2].

'gram-obsv'
Specifies the choice of frequency-weighted observability Grammian as follows:

'standard' Choice corresponding to standard Enns' method [1]. Default method.

'enhanced'
Choice corresponding to the stability enhanced modified Enns' method of [2].

'equil', *'scale'*
Boolean indicating whether equilibration (scaling) should be performed on *G* and *K* prior to order reduction. Default value is false if both `G.scaled == true, K.scaled == true` and true otherwise. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` function because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Algorithm
Uses SLICOT SB16AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.15.2 cfconred

`[Kr, info] = cfconred (G, F, L, ...)` [Function File]
`[Kr, info] = cfconred (G, F, L, ncr, ...)` [Function File]
`[Kr, info] = cfconred (G, F, L, opt, ...)` [Function File]
`[Kr, info] = cfconred (G, F, L, ncr, opt, ...)` [Function File]

Reduction of state-feedback-observer based controller by coprime factorization (CF). Given a plant *G*, state feedback gain *F* and full observer gain *L*, determine a reduced order controller *Kr*.

Inputs

G LTI model of the open-loop plant (A,B,C,D). It has m inputs, p outputs and n states.

F Stabilizing state feedback matrix (m-by-n).

L Stabilizing observer gain matrix (n-by-p).

ncr The desired order of the resulting reduced order controller *Kr*. If not specified, *ncr* is chosen automatically according to the description of key *'order'*.

... Optional pairs of keys and values. `"key1", value1, "key2", value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

Kr State-space model of reduced order controller.

info Struct containing additional information.

info.hsv The Hankel singular values of the extended system?!?. The *n* Hankel singular values are ordered decreasingly.

info.ncr The order of the obtained reduced order controller *Kr*.

Option Keys and Values

'order', 'ncr'
The desired order of the resulting reduced order controller *Kr*. If not specified, *ncr* is chosen automatically such that states with Hankel singular values *info.hsv* > *tol1* are retained.

'method' Order reduction approach to be used as follows:

'sr-bta', 'b'
Use the square-root Balance & Truncate method.

'bfsr-bta', 'f'
Use the balancing-free square-root Balance & Truncate method. Default method.

'sr-spa', 's'
Use the square-root Singular Perturbation Approximation method.

'bfsr-spa', 'p'
Use the balancing-free square-root Singular Perturbation Approximation method.

'cf' Specifies whether left or right coprime factorization is to be used as follows:

'left', 'l' Use left coprime factorization. Default method.

'right', 'r' Use right coprime factorization.

'feedback' Specifies whether *F* and *L* are fed back positively or negatively:

'+' A+BK and A+LC are both Hurwitz matrices.

'-' A-BK and A-LC are both Hurwitz matrices. Default value.

'tol1' If *'order'* is not specified, *tol1* contains the tolerance for determining the order of the reduced system. For model reduction, the recommended value of *tol1* is c*info.hsv(1), where c lies in the interval [0.00001, 0.001]. Default value is n*eps*info.hsv(1). If *'order'* is specified, the value of *tol1* is ignored.

'tol2' The tolerance for determining the order of a minimal realization of the coprime factorization controller. TOL2 <= TOL1. If not specified, n*eps*info.hsv(1) is chosen.

'equil', 'scale'
Boolean indicating whether equilibration (scaling) should be performed on system *G* prior to order reduction. Default value is true if `G.scaled == false` and false if `G.scaled == true`. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` function because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Algorithm
Uses SLICOT SB16BD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.15.3 fwcfconred

`[Kr, info] = fwcfconred (G, F, L, ...)` [Function File]
`[Kr, info] = fwcfconred (G, F, L, ncr, ...)` [Function File]
`[Kr, info] = fwcfconred (G, F, L, opt, ...)` [Function File]
`[Kr, info] = fwcfconred (G, F, L, ncr, opt, ...)` [Function File]

Reduction of state-feedback-observer based controller by frequency-weighted coprime factorization (FW CF). Given a plant *G*, state feedback gain *F* and full observer gain *L*, determine a reduced order controller *Kr* by using stability enforcing frequency weights.

Inputs

G LTI model of the open-loop plant (A,B,C,D). It has m inputs, p outputs and n states.

F Stabilizing state feedback matrix (m-by-n).

L Stabilizing observer gain matrix (n-by-p).

ncr The desired order of the resulting reduced order controller *Kr*. If not specified, *ncr* is chosen automatically according to the description of key *'order'*.

... Optional pairs of keys and values. `"key1", value1, "key2", value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

Kr State-space model of reduced order controller.

info Struct containing additional information.

info.hsv The Hankel singular values of the extended system?!?. The *n* Hankel singular values are ordered decreasingly.

info.ncr The order of the obtained reduced order controller *Kr*.

Option Keys and Values

'order', 'ncr'
The desired order of the resulting reduced order controller *Kr*. If not specified, *ncr* is chosen automatically such that states with Hankel singular values *info.hsv* > *tol1* are retained.

'method' Order reduction approach to be used as follows:

'sr', 'b' Use the square-root Balance & Truncate method.

'bfsr', 'f' Use the balancing-free square-root Balance & Truncate method. Default method.

'cf' Specifies whether left or right coprime factorization is to be used as follows:

'left', 'l' Use left coprime factorization.

'right', 'r' Use right coprime factorization. Default method.

'feedback' Specifies whether F and L are fed back positively or negatively:

'+' A+BK and A+LC are both Hurwitz matrices.

'-' A-BK and A-LC are both Hurwitz matrices. Default value.

'tol1' If *'order'* is not specified, *tol1* contains the tolerance for determining the order of the reduced system. For model reduction, the recommended value of *tol1* is c*info.hsv(1), where c lies in the interval [0.00001, 0.001]. Default value is n*eps*info.hsv(1). If *'order'* is specified, the value of *tol1* is ignored.

Algorithm

Uses SLICOT SB16CD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.15.4 spaconred

`[Kr, info] = spaconred (G, K, ...)` [Function File]
`[Kr, info] = spaconred (G, K, ncr, ...)` [Function File]
`[Kr, info] = spaconred (G, K, opt, ...)` [Function File]
`[Kr, info] = spaconred (G, K, ncr, opt, ...)` [Function File]

Controller reduction by frequency-weighted Singular Perturbation Approximation (SPA). Given a plant G and a stabilizing controller K, determine a reduced order controller Kr such that the closed-loop system is stable and closed-loop performance is retained.

The algorithm tries to minimize the frequency-weighted error

$$||V\ (K - K_r)\ W||_\infty = \min$$

where V and W denote output and input weightings.

Inputs

G LTI model of the plant. It has m inputs, p outputs and n states.

K LTI model of the controller. It has p inputs, m outputs and nc states.

ncr The desired order of the resulting reduced order controller *Kr*. If not specified, *ncr* is chosen automatically according to the description of key *'order'*.

... Optional pairs of keys and values. `"key1", value1, "key2", value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

Kr State-space model of reduced order controller.

info Struct containing additional information.

info.ncr The order of the obtained reduced order controller *Kr*.

info.ncs The order of the alpha-stable part of original controller *K*.

info.hsvc The Hankel singular values of the alpha-stable part of K. The *ncs* Hankel singular values are ordered decreasingly.

Option Keys and Values

'order', 'ncr'
The desired order of the resulting reduced order controller *Kr*. If not specified, *ncr* is chosen automatically such that states with Hankel singular values *info.hsvc* > *tol1* are retained.

'method' Order reduction approach to be used as follows:

'sr', 's' Use the square-root Singular Perturbation Approximation method.

'bfsr', 'p' Use the balancing-free square-root Singular Perturbation Approximation method. Default method.

'weight' Specifies the type of frequency-weighting as follows:

'none' No weightings are used (V = I, W = I).

'left', 'output'
Use stability enforcing left (output) weighting

$$V = (I - GK)^{-1}G, \qquad W = I$$

'right', 'input'
Use stability enforcing right (input) weighting

$$V = I, \qquad W = (I - GK)^{-1}G$$

'both', 'performance'
Use stability and performance enforcing weightings

$$V = (I - GK)^{-1}G, \qquad W = (I - GK)^{-1}$$

Default value.

'feedback' Specifies whether K is a positive or negative feedback controller:

'+' Use positive feedback controller. Default value.

'-' Use negative feedback controller.

'alpha' Specifies the ALPHA-stability boundary for the eigenvalues of the state dynamics matrix *K.A*. For a continuous-time controller, ALPHA <= 0 is the boundary value for the real parts of eigenvalues, while for a discrete-time controller, 0 <= ALPHA <= 1 represents the boundary value for the moduli of eigenvalues. The ALPHA-stability domain does not include the boundary. Default value is 0 for continuous-time controllers and 1 for discrete-time controllers.

'tol1' If *'order'* is not specified, *tol1* contains the tolerance for determining the order of the reduced controller. For model reduction, the recommended value of *tol1* is c*info.hsvc(1), where c lies in the interval [0.00001, 0.001]. Default value is info.ncs*eps*info.hsvc(1). If *'order'* is specified, the value of *tol1* is ignored.

'tol2' The tolerance for determining the order of a minimal realization of the ALPHA-stable part of the given controller. TOL2 <= TOL1. If not specified, ncs*eps*info.hsvc(1) is chosen.

'gram-ctrb'
Specifies the choice of frequency-weighted controllability Grammian as follows:

'standard' Choice corresponding to standard Enns' method [1]. Default method.

'enhanced'
Choice corresponding to the stability enhanced modified Enns' method of [2].

'gram-obsv'
Specifies the choice of frequency-weighted observability Grammian as follows:

'standard' Choice corresponding to standard Enns' method [1]. Default method.

'enhanced'
Choice corresponding to the stability enhanced modified Enns' method of [2].

'equil', 'scale'
Boolean indicating whether equilibration (scaling) should be performed on G and K prior to order reduction. Default value is false if both `G.scaled == true, K.scaled == true` and true otherwise. Note that for MIMO models, proper scaling of both inputs and outputs is of utmost importance. The input and output scaling can **not** be done by the equilibration option or the `prescale` function because these functions perform state transformations only. Furthermore, signals should not be scaled simply to a certain range. For all inputs (or outputs), a certain change should be of the same importance for the model.

Algorithm
Uses SLICOT SB16AD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.16 Experimental Data Handling

2.16.1 @iddata/cat

dat = cat (*dim*, *dat1*, *dat2*, ...) [Function File]

Concatenate iddata sets along dimension *dim*.

Inputs

dim Dimension along which the concatenation takes place.

1 Concatenate samples. The samples are concatenated in the following way: `dat.y{e} = [dat1.y{e}; dat2.y{e}; ...] dat.u{e} = [dat1.u{e}; dat2.u{e}; ...]` where *e* denotes the experiment. The number of experiments, outputs and inputs must be equal for all datasets. Equivalent to `vertcat`.

2 Concatenate inputs and outputs. The outputs and inputs are concatenated in the following way: `dat.y{e} = [dat1.y{e}, dat2.y{e}, ...]` `dat.u{e} = [dat1.u{e}, dat2.u{e}, ...]` where *e* denotes the experiment. The number of experiments and samples must be equal for all datasets. Equivalent to `horzcat`.

3 Concatenate experiments. The experiments are concatenated in the following way: `dat.y = [dat1.y; dat2.y; ...] dat.u = [dat1.u; dat2.u; ...]` The number of outputs and inputs must be equal for all datasets. Equivalent to `merge`.

dat1, *dat2*, . . .
iddata sets to be concatenated.

Outputs

dat Concatenated iddata set.

See also: horzcat, merge, vertcat.

2.16.2 @iddata/detrend

`dat = detrend (dat)` [Function File]
`dat = detrend (dat, ord)` [Function File]

Detrend outputs and inputs of dataset *dat* by removing the best fit of a polynomial of order *ord*. If *ord* is not specified, default value 0 is taken. This corresponds to removing a constant.

2.16.3 @iddata/diff

`dat = diff (dat)` [Function File]
`dat = diff (dat, k)` [Function File]

Return *k*-th difference of outputs and inputs of dataset *dat*. If *k* is not specified, default value 1 is taken.

2.16.4 @iddata/fft

`dat = fft (dat)` [Function File]
`dat = fft (dat, n)` [Function File]

Transform iddata objects from time to frequency domain using a Fast Fourier Transform (FFT) algorithm.

Inputs

dat iddata set containing signals in time-domain.

n Length of the FFT transformations. If *n* does not match the signal length, the signals in *dat* are shortened or padded with zeros. *n* is a vector with as many elements as there are experiments in *dat* or a scalar with a common length for all experiments. If not specified, the signal lengths are taken as default values.

Outputs

dat iddata identification dataset in frequency-domain. In order to preserve signal power and noise level, the FFTs are normalized by dividing each transform by the square root of the signal length. The frequency values are distributed equally from 0 to the Nyquist frequency. The Nyquist frequency is only included for even signal lengths.

2.16.5 @iddata/filter

`dat = filter (dat, sys)` [Function File]
`dat = filter (dat, b, a)` [Function File]

Filter output and input signals of dataset *dat*. The filter is specified either by LTI system *sys* or by transfer function polynomials *b* and *a* as described in the help text of Octave's built-in filter function. Type `help filter` for more information.

Inputs

dat iddata identification dataset containing signals in time-domain.

sys LTI object containing the discrete-time filter.

b Numerator polynomial of the discrete-time filter. Must be a row vector containing the coefficients of the polynomial in ascending powers of z^-1.

a Denominator polynomial of the discrete-time filter. Must be a row vector containing the coefficients of the polynomial in ascending powers of z^-1.

Outputs

dat iddata identification dataset with filtered output and input signals.

2.16.6 @iddata/get

`get (dat)` [Function File]
`value = get (dat, 'key')` [Function File]
`[val1, val2, ...] = get (dat, 'key1', 'key2', ...)` [Function File]

Access key values of iddata objects. Type `get(dat)` to display a list of available keys.

2.16.7 @iddata/iddata

`dat = iddata (y)` [Function File]
`dat = iddata (y, u)` [Function File]
`dat = iddata (y, u, tsam, ...)` [Function File]
`dat = iddata (y, u, [], ...)` [Function File]

Create identification dataset of output and input signals.

Inputs

y Real matrix containing the output signal in time-domain. For a system with *p* outputs and *n* samples, *y* is a n-by-p matrix. For data from multiple experiments, *y* becomes a e-by-1 or 1-by-e cell vector of n(i)-by-p matrices, where *e* denotes the number of experiments and n(i) the individual number of samples for each experiment.

u Real matrix containing the input signal in time-domain. For a system with *m* inputs and *n* samples, *u* is a n-by-m matrix. For data from multiple experiments, *u* becomes a e-by-1 or 1-by-e cell vector of n(i)-by-m matrices, where *e* denotes the number of experiments and n(i) the individual number of samples for each experiment. If *u* is not specified or an empty element `[]` is passed, *dat* becomes a time series dataset.

tsam Sampling time. If not specified, default value -1 (unspecified) is taken. For multi-experiment data, *tsam* becomes a e-by-1 or 1-by-e cell vector containing individual sampling times for each experiment. If a scalar *tsam* is provided, then all experiments have the same sampling time.

... Optional pairs of properties and values.

Outputs

dat iddata identification dataset.

Option Keys and Values

'expname' The name of the experiments in *dat*. Cell vector of length e containing strings. Default names are `{'exp1', 'exp2', ...}`

'y' Output signals. See 'Inputs' for details.

'outname' The name of the output channels in *dat*. Cell vector of length p containing strings. Default names are `{'y1', 'y2', ...}`

'outunit' The units of the output channels in *dat*. Cell vector of length p containing strings.

'u' Input signals. See 'Inputs' for details.

'inname' The name of the input channels in *dat*. Cell vector of length m containing strings. Default names are `{'u1', 'u2', ...}`

'inunit' The units of the input channels in *dat*. Cell vector of length m containing strings.

'tsam' Sampling time. See 'Inputs' for details.

'timeunit' The units of the sampling times in *dat*. Cell vector of length e containing strings.

'name' String containing the name of the dataset.

'notes' String or cell of string containing comments.

'userdata' Any data type.

2.16.8 @iddata/ifft

`dat = ifft (dat)` [Function File]

Transform iddata objects from frequency to time domain.

Inputs

dat iddata set containing signals in frequency domain. The frequency values must be distributed equally from 0 to the Nyquist frequency. The Nyquist frequency is only included for even signal lengths.

Outputs

dat iddata identification dataset in time domain. In order to preserve signal power and noise level, the FFTs are normalized by multiplying each transform by the square root of the signal length.

2.16.9 @iddata/merge

`dat = merge (dat1, dat2, ...)` [Function File]

Concatenate experiments of iddata datasets. The experiments are concatenated in the following way: `dat.y = [dat1.y; dat2.y; ...]` `dat.u = [dat1.u; dat2.u; ...]` The number of outputs and inputs must be equal for all datasets.

2.16.10 @iddata/nkshift

`dat = nkshift (dat, nk)` [Function File]

`dat = nkshift (dat, nk, 'append')` [Function File]

Shift input channels of dataset *dat* according to integer *nk*. A positive value of *nk* means that the input channels are delayed *nk* samples. By default, both input and output signals are shortened by *nk* samples. If a third argument *'append'* is passed, the output signals are left untouched while *nk* zeros are appended to the (shortened) input signals such that the number of samples in *dat* remains constant.

2.16.11 @iddata/plot

`plot (dat)` [Function File]

`plot (dat, exp)` [Function File]

Plot signals of iddata identification datasets on the screen. The signals are plotted experiment-wise, either in time- or frequency-domain. For multi-experiment datasets, press any key to switch to the next experiment. If the plot of a single experiment should be saved by the `print` command, use `plot(dat,exp)`, where *exp* denotes the desired experiment.

2.16.12 @iddata/resample

dat = resample (*dat*, *p*, *q*) [Function File]
dat = resample (*dat*, *p*, *q*, *n*) [Function File]
dat = resample (*dat*, *p*, *q*, *h*) [Function File]

Change the sample rate of the output and input signals in dataset *dat* by a factor of `p/q`. This is performed using a polyphase algorithm. The anti-aliasing FIR filter can be specified as follows: Either by order *n* (scalar) with default value 0. The band edges are then chosen automatically. Or by impulse response *h* (vector). Requires the signal package to be installed.

Algorithm

Uses functions `fir1` and `resample` from the signal package.

References

1. J. G. Proakis and D. G. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, 4th ed., Prentice Hall, 2007. Chap. 6
2. A. V. Oppenheim, R. W. Schafer and J. R. Buck, Discrete-time signal processing, Signal processing series, Prentice-Hall, 1999

2.16.13 @iddata/set

set (*dat*) [Function File]
set (*dat*, *'key'*, *value*, ...) [Function File]
dat = set (*dat*, *'key'*, *value*, ...) [Function File]

Set or modify keys of iddata objects. If no return argument *dat* is specified, the modified IDDATA object is stored in input argument *dat*. `set` can handle multiple keys in one call: `set (dat, 'key1', val1, 'key2', val2, 'key3', val3)`. `set (dat)` prints a list of the object's key names.

2.16.14 @iddata/size

nvec = size (*dat*) [Function File]
ndim = size (*dat*, *dim*) [Function File]
[*n*, *p*, *m*, *e*] = size (*dat*) [Function File]

Return dimensions of iddata set *dat*.

Inputs

dat iddata set.

dim If given a second argument, `size` will return the size of the corresponding dimension.

Outputs

nvec Row vector. The first element is the total number of samples (rows of dat.y and dat.u). The second element is the number of outputs (columns of dat.y) and the third element the number of inputs (columns of dat.u). The fourth element is the number of experiments.

ndim Scalar value. The size of the dimension *dim*.

n Row vector containing the number of samples of each experiment.

p Number of outputs.

m Number of inputs.

e Number of experiments.

2.17 System Identification

2.17.1 arx

`[sys, x0] = arx (dat, n, ...)` [Function File]
`[sys, x0] = arx (dat, n, opt, ...)` [Function File]
`[sys, x0] = arx (dat, opt, ...)` [Function File]
`[sys, x0] = arx (dat, 'na', na, 'nb', nb)` [Function File]

Estimate ARX model using QR factorization.

$$A(q)\,y(t) = B(q)\,u(t)\;+\;e(t)$$

Inputs

dat iddata identification dataset containing the measurements, i.e. time-domain signals.

n The desired order of the resulting model *sys*.

... Optional pairs of keys and values. `'key1', value1, 'key2', value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

sys Discrete-time transfer function model. If the second output argument *x0* is returned, *sys* becomes a state-space model.

x0 Initial state vector. If *dat* is a multi-experiment dataset, *x0* becomes a cell vector containing an initial state vector for each experiment.

Option Keys and Values

'na' Order of the polynomial A(q) and number of poles.

'nb' Order of the polynomial B(q)+1 and number of zeros+1. *nb* <= *na*.

'nk' Input-output delay specified as number of sampling instants. Scalar positive integer. This corresponds to a call to function `nkshift`, followed by padding the B polynomial with *nk* leading zeros.

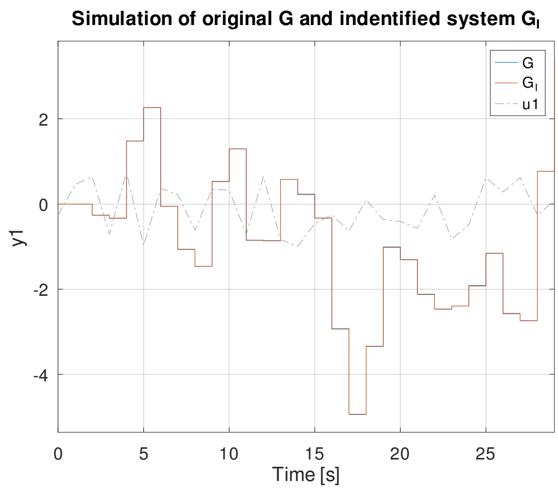
Algorithm
Uses the formulae given in [1] on pages 318-319, 'Solving for the LS Estimate by QR Factorization'. Uses SLICOT IB01CD for initial conditions, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

References

1. Ljung, L. (1999) *System Identification: Theory for the User: Second Edition*. Prentice Hall, New Jersey, USA.

Example: 1

```
T = 1;
N = 30;
G = tf ([1  2 ], [1 -1 1 -0.5], T);
u = 2*rand(N,1) - 1;
e = 0.01*randn (N,1);
y = lsim (G, u) + e;
dat = iddata (y, u, T);
G_I = arx (dat, 3);  # filter form (poly in z^(-1), inputs u and e)
G_I = G_I(1,1);
close (clf());
lsim (G, G_I, u);
title ("Simulation of original G and indentified system G_I")
```



2.17.2 moen4

[*sys*, *x0*, *info*] = moen4 (*dat*, ...) [Function File]
[*sys*, *x0*, *info*] = moen4 (*dat*, *n*, ...) [Function File]
[*sys*, *x0*, *info*] = moen4 (*dat*, *opt*, ...) [Function File]
[*sys*, *x0*, *info*] = moen4 (*dat*, *n*, *opt*, ...) [Function File]

Estimate state-space model using combined subspace method: MOESP algorithm for finding the matrices A and C, and N4SID algorithm for finding the matrices B and D. If no output arguments are given, the singular values are plotted on the screen in order to estimate the system order.

Inputs

dat iddata set containing the measurements, i.e. time-domain signals.

n The desired order of the resulting state-space system *sys*. If not specified, *n* is chosen automatically according to the singular values and tolerances.

... Optional pairs of keys and values. `'key1', value1, 'key2', value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

sys Discrete-time state-space model.

x0 Initial state vector. If *dat* is a multi-experiment dataset, *x0* becomes a cell vector containing an initial state vector for each experiment.

info Struct containing additional information.

- *info.K* Kalman gain matrix.
- *info.Q* State covariance matrix.
- *info.Ry* Output covariance matrix.
- *info.S* State-output cross-covariance matrix.
- *info.L* Noise variance matrix factor. LL'=Ry.

Option Keys and Values

'n' The desired order of the resulting state-space system *sys*. $s > n > 0$.

's' The number of block rows s in the input and output block Hankel matrices to be processed. $s > 0$. In the MOESP theory, s should be larger than n, the estimated dimension of state vector.

'alg', 'algorithm'
Specifies the algorithm for computing the triangular factor R, as follows:

- *'C'* Cholesky algorithm applied to the correlation matrix of the input-output data. Default method.
- *'F'* Fast QR algorithm.
- *'Q'* QR algorithm applied to the concatenated block Hankel matrices.

'tol' Absolute tolerance used for determining an estimate of the system order. If *tol* >= 0, the estimate is indicated by the index of the last singular value greater than or equal to *tol*. (Singular values less than *tol* are considered as zero.) When *tol* = 0, an internally computed default value, *tol* = *s***eps**SV(1), is used, where SV(1) is the maximal singular value, and *eps* is the relative machine precision. When *tol* < 0, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default value is 0.

'rcond' The tolerance to be used for estimating the rank of matrices. If the user sets *rcond* > 0, the given value of *rcond* is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/*rcond* is considered to be of full rank. If the user sets *rcond* <= 0, then an implicitly computed, default tolerance, defined by *rcond* = m*n**eps*, is used instead, where *eps* is the relative machine precision. Default value is 0.

'confirm' Specifies whether or not the user's confirmation of the system order estimate is desired, as follows:

- *true* User's confirmation.
- *false* No confirmation. Default value.

'noiseinput'
The desired type of noise input channels.

- *'n'* No error inputs. Default value.

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

- *'e'* Return *sys* as a (p-by-m+p) state-space model with both measured input channels u and noise channels e with covariance matrix *Ry*.

$$x_{k+1} = Ax_k + Bu_k + Ke_k$$

$$y_k = Cx_k + Du_k + e_k$$

'v' Return *sys* as a (p-by-m+p) state-space model with both measured input channels u and white noise channels v with identity covariance matrix.

$$x_{k+1} = Ax_k + Bu_k + KLv_k$$

$$y_k = Cx_k + Du_k + Lv_k$$
$$e = Lv, \;\; LL^T = R_y$$

'k' Return *sys* as a Kalman predictor for simulation.

$$\widehat{x}_{k+1} = A\widehat{x}_k + Bu_k + K(y_k - \widehat{y}_k)$$

$$\widehat{y}_k = C\widehat{x}_k + Du_k$$

$$\widehat{x}_{k+1} = (A - KC)\widehat{x}_k + (B - KD)u_k + Ky_k$$

$$\widehat{y}_k = C\widehat{x}_k + Du_k + 0y_k$$

Algorithm

Uses IB01BD and IB01CD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.17.3 moesp

`[sys, x0, info] = moesp (dat, ...)` [Function File]
`[sys, x0, info] = moesp (dat, n, ...)` [Function File]
`[sys, x0, info] = moesp (dat, opt, ...)` [Function File]
`[sys, x0, info] = moesp (dat, n, opt, ...)` [Function File]

Estimate state-space model using MOESP algorithm. MOESP: Multivariable Output Error State sPace. If no output arguments are given, the singular values are plotted on the screen in order to estimate the system order.

Inputs

dat iddata set containing the measurements, i.e. time-domain signals.

n The desired order of the resulting state-space system *sys*. If not specified, *n* is chosen automatically according to the singular values and tolerances.

... Optional pairs of keys and values. `'key1', value1, 'key2', value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

sys Discrete-time state-space model.

x0 Initial state vector. If *dat* is a multi-experiment dataset, *x0* becomes a cell vector containing an initial state vector for each experiment.

info Struct containing additional information.

info.K Kalman gain matrix.

info.Q State covariance matrix.

info.Ry Output covariance matrix.

info.S State-output cross-covariance matrix.

info.L Noise variance matrix factor. LL'=Ry.

Option Keys and Values

'n' The desired order of the resulting state-space system *sys*. $s > n > 0$.

's' The number of block rows s in the input and output block Hankel matrices to be processed. $s > 0$. In the MOESP theory, s should be larger than n, the estimated dimension of state vector.

'alg', 'algorithm'
Specifies the algorithm for computing the triangular factor R, as follows:

- *'C'* Cholesky algorithm applied to the correlation matrix of the input-output data. Default method.
- *'F'* Fast QR algorithm.
- *'Q'* QR algorithm applied to the concatenated block Hankel matrices.

'tol' Absolute tolerance used for determining an estimate of the system order. If *tol* >= 0, the estimate is indicated by the index of the last singular value greater than or equal to *tol*. (Singular values less than *tol* are considered as zero.) When *tol* = 0, an internally computed default value, *tol* = s**eps**SV(1), is used, where SV(1) is the maximal singular value, and *eps* is the relative machine precision. When *tol* < 0, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default value is 0.

'rcond' The tolerance to be used for estimating the rank of matrices. If the user sets *rcond* > 0, the given value of *rcond* is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/*rcond* is considered to be of full rank. If the user sets *rcond* <= 0, then an implicitly computed, default tolerance, defined by *rcond* = m*n**eps*, is used instead, where *eps* is the relative machine precision. Default value is 0.

'confirm' Specifies whether or not the user's confirmation of the system order estimate is desired, as follows:

- *true* User's confirmation.
- *false* No confirmation. Default value.

'noiseinput'
The desired type of noise input channels.

- *'n'* No error inputs. Default value.

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

- *'e'* Return *sys* as a (p-by-m+p) state-space model with both measured input channels u and noise channels e with covariance matrix *Ry*.

$$x_{k+1} = Ax_k + Bu_k + Ke_k$$

$$y_k = Cx_k + Du_k + e_k$$

'v' Return *sys* as a (p-by-m+p) state-space model with both measured input channels u and white noise channels v with identity covariance matrix.

$$x_{k+1} = Ax_k + Bu_k + KLv_k$$

$$y_k = Cx_k + Du_k + Lv_k$$
$$e = Lv, \ LL^T = R_y$$

'k' Return *sys* as a Kalman predictor for simulation.

$$\widehat{x}_{k+1} = A\widehat{x}_k + Bu_k + K(y_k - \widehat{y}_k)$$

$$\widehat{y}_k = C\widehat{x}_k + Du_k$$

$$\widehat{x}_{k+1} = (A - KC)\widehat{x}_k + (B - KD)u_k + Ky_k$$

$$\widehat{y}_k = C\widehat{x}_k + Du_k + 0y_k$$

Algorithm

Uses IB01BD and IB01CD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.17.4 n4sid

[*sys*, *x0*, *info*] = n4sid (*dat*, ...) [Function File]
[*sys*, *x0*, *info*] = n4sid (*dat*, *n*, ...) [Function File]
[*sys*, *x0*, *info*] = n4sid (*dat*, *opt*, ...) [Function File]
[*sys*, *x0*, *info*] = n4sid (*dat*, *n*, *opt*, ...) [Function File]

Estimate state-space model using N4SID algorithm. N4SID: Numerical algorithm for Subspace State Space System IDentification. If no output arguments are given, the singular values are plotted on the screen in order to estimate the system order.

Inputs

dat iddata set containing the measurements, i.e. time-domain signals.

n The desired order of the resulting state-space system *sys*. If not specified, *n* is chosen automatically according to the singular values and tolerances.

... Optional pairs of keys and values. `'key1', value1, 'key2', value2`.

opt Optional struct with keys as field names. Struct *opt* can be created directly or by function `options`. `opt.key1 = value1, opt.key2 = value2`.

Outputs

sys Discrete-time state-space model.

x0 Initial state vector. If *dat* is a multi-experiment dataset, *x0* becomes a cell vector containing an initial state vector for each experiment.

info Struct containing additional information.

info.K Kalman gain matrix.

info.Q State covariance matrix.

info.Ry Output covariance matrix.

info.S State-output cross-covariance matrix.

info.L Noise variance matrix factor. LL'=Ry.

Option Keys and Values

'n' The desired order of the resulting state-space system *sys*. $s > n > 0$.

's' The number of block rows *s* in the input and output block Hankel matrices to be processed. $s > 0$. In the MOESP theory, *s* should be larger than *n*, the estimated dimension of state vector.

'alg', 'algorithm'
Specifies the algorithm for computing the triangular factor R, as follows:

'C' Cholesky algorithm applied to the correlation matrix of the input-output data. Default method.

'F' Fast QR algorithm.

'Q' QR algorithm applied to the concatenated block Hankel matrices.

'tol' Absolute tolerance used for determining an estimate of the system order. If *tol* >= 0, the estimate is indicated by the index of the last singular value greater than or equal to *tol*. (Singular values less than *tol* are considered as zero.) When *tol* = 0, an internally computed default value, *tol* = *s***eps**SV(1), is used, where SV(1) is the maximal singular value, and *eps* is the relative machine precision. When *tol* < 0, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default value is 0.

'rcond' The tolerance to be used for estimating the rank of matrices. If the user sets *rcond* > 0, the given value of *rcond* is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/*rcond* is considered to be of full rank. If the user sets *rcond* <= 0, then an implicitly computed, default tolerance, defined by *rcond* = m*n**eps*, is used instead, where *eps* is the relative machine precision. Default value is 0.

'confirm' Specifies whether or not the user's confirmation of the system order estimate is desired, as follows:

true User's confirmation.

false No confirmation. Default value.

'noiseinput'
The desired type of noise input channels.

'n' No error inputs. Default value.

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

'e' Return *sys* as a (p-by-m+p) state-space model with both measured input channels u and noise channels e with covariance matrix *Ry*.

$$x_{k+1} = Ax_k + Bu_k + Ke_k$$

$$y_k = Cx_k + Du_k + e_k$$

'v' Return *sys* as a (p-by-m+p) state-space model with both measured input channels u and white noise channels v with identity covariance matrix.

$$x_{k+1} = Ax_k + Bu_k + KLv_k$$

$$y_k = Cx_k + Du_k + Lv_k$$

$$e = Lv, \; LL^T = R_y$$

'k' Return *sys* as a Kalman predictor for simulation.

$$\widehat{x}_{k+1} = A\widehat{x}_k + Bu_k + K(y_k - \widehat{y}_k)$$

$$\widehat{y}_k = C\widehat{x}_k + Du_k$$

$$\widehat{x}_{k+1} = (A - KC)\widehat{x}_k + (B - KD)u_k + Ky_k$$

$$\widehat{y}_k = C\widehat{x}_k + Du_k + 0y_k$$

Algorithm

Uses IB01BD and IB01CD, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer).

2.18 Overloaded LTI Operators

2.18.1 @lti/ctranspose

`SYST = ctranspose (SYS)` [Function File]

Conjugate transpose or pertransposition of LTI objects. Used by Octave for "sys'". For a transfer-function matrix G, G' denotes the conjugate of G given by G.'(-s) for a continuous-time system or G.'(1/z) for a discrete-time system. The frequency response of the pertransposition of G is the Hermitian (conjugate) transpose of G(jw), i.e. freqresp (G', w) = freqresp (G, w)'. **WARNING:** Do **NOT** use this for dual problems, use the transpose "sys.'" (note the dot) instead.

Inputs

SYS System to be transposed.

Outputs

SYST Conjugate transposed of *SYS*.

2.18.2 @lti/end

`end` [Overloaded Operator]

End indexing for LTI objects. Used by Octave for "sys(1:end, end-1)".

2.18.3 @lti/horzcat

`horzcat` [Overloaded Operator]

Horizontal concatenation of LTI objects. If necessary, object conversion is done by sys_group. Used by Octave for "[sys1, sys2]".

2.18.4 @lti/inv

SYSI = inv (*SYS*) [Function File]

Inversion of LTI objects.

Inputs

SYS System to be inverted.

Outputs

SYSI Inverteted system of *SYS*.

2.18.5 @lti/minus

minus [Overloaded Operator]

Binary subtraction of LTI objects. If necessary, object conversion is done by sys_group. Used by Octave for "sys1 - sys2".

2.18.6 @lti/mldivide

mldivide [Overloaded Operator]

Matrix left division of LTI objects. If necessary, object conversion is done by sys_group in mtimes. Used by Octave for "sys1 \\ sys2".

2.18.7 @lti/mpower

SYSP = inv (*SYS*, *E*) [Function File]

Matrix power of LTI objects. The exponent must be an integer. Used by Octave for "sys^int".

Inputs

SYS System for which the power by exponent *E* has to be calculated.

E Exponent (integer).

Outputs

SYSP Resulting power of *SYS* by *E*.

2.18.8 @lti/mrdivide

mrdivide [Overloaded Operator]

Matrix right division of LTI objects. If necessary, object conversion is done by sys_group in mtimes. Used by Octave for "sys1 / sys2".

2.18.9 @lti/mtimes

mtimes [Overloaded Operator]

Matrix multiplication of LTI objects. If necessary, object conversion is done by sys_group. Used by Octave for "sys1 * sys2".

2.18.10 @lti/plus

plus [Overloaded Operator]

Binary addition of LTI objects. If necessary, object conversion is done by sys_group. Used by Octave for "sys1 + sys2". Operation is also known as "parallel connection".

2.18.11 @lti/repmat

rsys = repmat (*sys*, *m*, *n*) [Function File]
rsys = repmat (*sys*, [*m*, *n*]) [Function File]
rsys = repmat (*sys*, *m*) [Function File]

Form a block transfer matrix of *sys* with *m* copies vertically and *n* copies horizontally. If *n* is not specified, it is set to *m*. `repmat (sys, 2, 3)` is equivalent to `[sys, sys, sys; sys, sys, sys]`.

2.18.12 @lti/subsasgn

subsasgn [Overloaded Operator]

Subscripted assignment for LTI objects. Used by Octave for "sys.property = value".

2.18.13 @lti/subsref

subsref [Overloaded Operator]

Subscripted reference for LTI objects. Used by Octave for "sys = sys(2:4, :)" or "val = sys.prop".

2.18.14 @lti/times

times [Overloaded Operator]

Hadamard/Schur product of transfer function matrices. Also known as element-wise multiplication. Used by Octave for "sys1 .* sys2".

Example

```
# Compute Relative-Gain Array
G = tf (Boeing707)
RGA = G .* inv (G).'
# Gain at 0 rad/s
RGA(0)
```

2.18.15 @lti/transpose

transpose [Overloaded Operator]

Transpose of LTI objects. Used by Octave for "sys.'". Useful for dual problems, i.e. controllability and observability or designing estimator gains with `lqr` and `place`.

2.18.16 @lti/uminus

uminus [Overloaded Operator]

Unary minus of LTI object. Used by Octave for "-sys".

2.18.17 @lti/uplus

uplus [Overloaded Operator]

Unary plus of LTI object. Used by Octave for "+sys".

2.18.18 @lti/vertcat

vertcat [Overloaded Operator]

Vertical concatenation of LTI objects. If necessary, object conversion is done by sys_group. Used by Octave for "[sys1; sys2]".

2.19 Overloaded IDDATA Operators

2.19.1 @iddata/end

`end` [Overloaded Operator]

End indexing for IDDATA objects. Used by Octave for "dat(1:end)".

2.19.2 @iddata/horzcat

dat = `horzcat` (*dat1*, *dat2*, ...) [Function File]

Horizontal concatenation of iddata datasets.

The outputs and inputs are concatenated in the following way: `dat.y{e} = [dat1.y{e}, dat2.y{e}, ...] dat.u{e} = [dat1.u{e}, dat2.u{e}, ...]` where *e* denotes the experiment. The number of experiments and samples must be equal for all datasets.

2.19.3 @iddata/subsasgn

`subsasgn` [Overloaded Operator]

Subscripted assignment for iddata objects. Used by Octave for "dat.property = value".

2.19.4 @iddata/subsref

`subsref` [Overloaded Operator]

Subscripted reference for iddata objects. Used by Octave for "dat = dat(2:4, :)" or "val = dat.prop".

2.19.5 @iddata/vertcat

dat = `vertcat` (*dat1*, *dat2*, ...) [Function File]

Vertical concatenation of iddata datasets. The samples are concatenated in the following way: `dat.y{e} = [dat1.y{e}; dat2.y{e}; ...] dat.u{e} = [dat1.u{e}; dat2.u{e}; ...]` where *e* denotes the experiment. The number of experiments, outputs and inputs must be equal for all datasets.

2.20 Examples

2.20.1 Anderson

`Anderson` [Function File]

Frequency-weighted coprime factorization controller reduction [1].

References

1. Anderson, B.D.O.: *Controller Reduction: Concepts and Approaches*, IEEE Transactions of Automatic Control, Vol. 34, No. 8, August 1989.

2.20.2 BMWengine

sys = `BMWengine` [Function File]
sys = `BMWengine ("`*scaled*`")` [Function File]
sys = `BMWengine ("`*unscaled*`")` [Function File]

Model of the BMW 4-cylinder engine at ETH Zurich's control laboratory.

```
OPERATING POINT
Drosselklappenstellung     alpha_DK = 10.3 Grad
Saugrohrdruck              p_s = 0.48 bar
Motordrehzahl              n = 860 U/min
Lambda-Messwert            lambda = 1.000
Relativer Wandfilminhalt   nu = 1

INPUTS
U_1 Sollsignal Drosselklappenstellung   [Grad]
U_2 Relative Einspritzmenge             [-]
U_3 Zuendzeitpunkt                      [Grad KW]
M_L Lastdrehmoment                      [Nm]

STATES
X_1 Drosselklappenstellung     [Grad]
X_2 Saugrohrdruck              [bar]
X_3 Motordrehzahl              [U/min]
X_4 Messwert Lamba-Sonde       [-]
X_5 Relativer Wandfilminhalt   [-]

OUTPUTS
Y_1 Motordrehzahl              [U/min]
Y_2 Messwert Lambda-Sonde      [-]

SCALING
U_1N, X_1N   1 Grad
U_2N, X_4N, X_5N, Y_2N   0.05
U_3N   1.6 Grad KW
X_2N   0.05 bar
X_3N, Y_1N   200 U/min
```

2.20.3 Boeing707

sys = `Boeing707` [Function File]

Creates a linearized state-space model of a Boeing 707-321 aircraft at $v = 80\,\frac{m}{s}$ $(M = 0.26$, $G_{a0} = -3°$, $\alpha_0 = 4°$, $\kappa = 50°)$.

System inputs:

1. Thrust
2. Elevator angle

System outputs:

1. Airspeed
2. Pitch angle

Reference: R. Brockhaus: *Flugregelung* (Flight Control), Springer, 1994.

2.20.4 MDSSystem

`MDSSystem` [Function File]

Run the example for robust control of a mass-damper-spring system.

Run `edit MDSSystem` to open the example file.

2.20.5 Madievski

`Madievski` [Function File]

Demonstration of frequency-weighted controller reduction.

The system considered in this example has been studied by Madievski and Anderson [1] and comprises four spinning disks. The disks are connected by a flexible rod, a motor applies torque to the third disk, and the angular displacement of the first disk is the variable of interest. The state-space model of eighth order is non-minimumphase and unstable. The continuous-time LQG controller used in [1] is open-loop stable and of eighth order like the plant. This eighth-order controller shall be reduced by frequency-weighted singular perturbation approximation (SPA). The major aim of this reduction is the preservation of the closed-loop transfer function. This means that the error in approximation of the controller K by the reduced-order controller Kr is minimized by

$$\min_{K_r} \; ||W \; (K - K_r) \; V||_\infty$$

where weights W and V are dictated by the requirement to preserve (as far as possible) the closed-loop transfer function. In minimizing the error, they cause the approximation process for K to be more accurate at certain frequencies. Suggested by [1] is the use of the following stability and performance enforcing weights:

$$W = (I - GK)^{-1} G, \qquad V = (I - GK)^{-1}$$

This example script reduces the eighth-order controller to orders four and two by the function call `Kr = spaconred (G, K, nr, 'feedback', '-')` where argument *nr* denotes the desired order (4 or 2). The key-value pair `'feedback', '-'` allows the reduction of negative feedback controllers while the default setting expects positive feedback controllers. The frequency responses of the original and reduced-order controllers are depicted in figure 1, the step responses of the closed loop in figure 2. There is no visible difference between the step responses of the closed-loop systems with original (blue) and fourth order (green) controllers. The second order controller (red) causes ripples in the step response, but otherwise the behavior of the system is unaltered. This leads to the conclusion that function `spaconred` is well suited to reduce the order of controllers considerably, while stability and performance are retained.

Reference

1. Madievski, A.G. and Anderson, B.D.O. *Sampled-Data Controller Reduction Procedure*, IEEE Transactions of Automatic Control, Vol. 40, No. 11, November 1995

2.20.6 VLFamp

`VLFamp` [Function File]

result = `VLFamp` (*verbose*) [Function File]

Calculations on a two stage preamp for a multi-turn, air-core solenoid loop antenna for the reception of signals below 30kHz.

The Octave Control Package functions are used extensively to approximate the behavior of operational amplifiers and passive electrical circuit elements.

This example presents several 'screen' pages of documentation of the calculations and some reasoning about why. Plots of the results are presented in most cases.

The process is to display a 'screen' page of text followed by the calculation and a 'Press return to continue' message. To proceed in the example, press return. ^C to exit.

At one point in the calculations, the process may seem to hang, but, this is because of extensive calculations.

The returned transfer function is more than 100 characters long so will wrap in screens that are narrow and appear jumbled.

2.20.7 WestlandLynx

sys = `WestlandLynx` [Function File]

Model of the Westland Lynx Helicopter about hover.

System variables

System inputs

```
main rotor collective
longitudinal cyclic
lateral cyclic
tail rotor collective
```

System states

```
pitch attitude           theta      [rad]
roll attitude            phi        [rad]
roll rate (body-axis)    p          [rad/s]
pitch rate (body-axis)   q          [rad/s]
yaw rate                 xi         [rad/s]
forward velocity         v_x        [ft/s]
lateral velocity         v_y        [ft/s]
vertical velocity        v_z        [ft/s]
```

System outputs

```
heave velocity           H_dot      [ft/s]
pitch attitude           theta      [rad]
roll attitude            phi        [rad]
heading rate             psi_dot    [rad/s]
roll rate                p          [rad/s]
pitch rate               q          [rad/s]
```

References

1. Skogestad, S. and Postlethwaite I. (2005) *Multivariable Feedback Control: Analysis and Design: Second Edition*. Wiley. http://www.nt.ntnu.no/users/skoge/book/2nd_edition/matlab_m/matfiles.html

2.20.8 optiPID

`optiPID` [Function File]

Numerical optimization of a PID controller using an objective function.

The objective function is located in the file `optiPIDfun`. Type `which optiPID` to locate, `edit optiPID` to open and simply `optiPID` to run the example file. In this example called `optiPID`, loosely based on [1], it is assumed that the plant

$$P(s) = \frac{1}{(s^2 + s + 1)(s + 1)^4}$$

is controlled by a PID controller with second-order roll-off

$$C(s) = K_P(1 + \frac{1}{T_I s} + T_D s)\frac{1}{(\tau s + 1)^2}$$

in the usual negative feedback structure

$$T(s) = \frac{L(s)}{1 + L(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)}$$

The plant P(s) is of higher order but benign. The initial values for the controller parameters K_P, T_I and T_D are obtained by applying the Astroem and Haegglund rules [2]. These values are to be improved using a numerical optimization as shown below. As with all numerical methods, this approach can never guarantee that a proposed solution is a global minimum. Therefore, good initial guesses for the parameters to be optimized are very important. The Octave function `fminsearch` minimizes the objective function J, which is chosen to be

$$J(K_P, T_I, T_D) = \mu_1 \int_0^{\infty} t|e(t)|dt + \mu_2(||y(t)||_\infty - 1) + \mu_3||S(jw)||_\infty$$

This particular objective function penalizes the integral of time-weighted absolute error

$$ITAE = \int_0^{\infty} t|e(t)|dt$$

and the maximum overshoot

$$y_{max} - 1 = ||y(t)||_\infty - 1$$

to a unity reference step $r(t) = \varepsilon(t)$ in the time domain. In the frequency domain, the sensitivity $M_s = ||S(jw)||_\infty$ is minimized for good robustness, where S(s) denotes the *sensitivity* transfer function

$$S(s) = \frac{1}{1 + L(s)} = \frac{1}{1 + P(s)\, C(s)}$$

The constants μ_1, μ_2 and μ_3 are *relative weighting factors* or «tuning knobs» which reflect the importance of the different design goals. Varying these factors corresponds to changing the emphasis from, say, high performance to good robustness. The main advantage of this approach is the possibility to explore the tradeoffs of the design problem in a systematic way. In a first approach, all three design objectives are weigthed equally. In subsequent iterations, the parameters $\mu_1 = 1$, $\mu_2 = 10$ and $\mu_3 = 20$ are found to yield satisfactory closed-loop performance. This controller results in a system with virtually no overshoot and a phase margin of 64 degrees.

References

1. Guzzella, L. *Analysis and Design of SISO Control Systems*, VDF Hochschulverlag, ETH Zurich, 2007

2. Astroem, K. and Haegglund, T. *PID Controllers: Theory, Design and Tuning*, Second Edition, Instrument Society of America, 1995

2.21 Miscellaneous

2.21.1 @ss/display

`display` (*SYS*) [Function File]

Display routine for SS objects.

Inputs

SYS System to be displayed.

2.21.2 db2mag

mag = `db2mag` (*db*) [Function File]

Convert Decibels (dB) to Magnitude.

Inputs

db Decibel (dB) value(s). Both real-valued scalars and matrices are accepted.

Outputs

mag Magnitude value(s).

See also: mag2db.

2.21.3 doc_control

`doc_control` *fcn1 fcn2* ...
st = `doc_control` (*fcn1*, *fcn2*, ...)

Open online documentation of the Control package in the system's standard browser.

Note: Since Octave 11 and Control Package 4.2.0 the package documentation is automatically shown in the documentation browser of the Octave GUI as soon as the Control Package is loaded. Help to all package functions can then also be accessed by the comamnd `doc ...`. However, this function is still useful when using Octave in the command line.

Inputs

fcn1, fcn2, ...
Function names for which the documentation should be displayed. If no function name is given, the index of the documentation is shown. If one of the function names is 'license', copyright and license information are displayed.

Outputs

st The return value *st* has one of the values:
- '`0`' Found and opened system browser successfully.
- '`1`' System browser not found.
- '`2`' System browser found, but an error occurred.

2.21.4 mag2db

db = `mag2db` (*mag*) [Function File]

Convert Magnitude to Decibels (dB).

Inputs

mag Magnitude value(s). Both real-valued scalars and matrices are accepted.

Outputs

db Decibel (dB) value(s).

See also: db2mag.

2.21.5 options

opt = `options` (*'key1'*, *value1*, *'key2'*, *value2*, ...) [Function File]

Create options struct *opt* from a number of key and value pairs. For use with order reduction and system identification functions. Option structs are a way to avoid typing the same key and value pairs over and over again.

Inputs

key, property
The name of the property.

value The value of the property.

Outputs

opt Struct with fields for each key.

Example

```
octave:1> opt = options ("method", "spa", "tol", 1e-6)
opt =

  scalar structure containing the fields:

    method = spa
    tol =  1.0000e-06

octave:2> save filename opt
octave:3> # save the struct 'opt' to file 'filename' for later use
octave:4> load filename
octave:5> # load struct 'opt' from file 'filename'
```

2.21.6 repsys

rsys = repsys (*sys*, *m*, *n*) [Function File]
rsys = repsys (*sys*, [*m*, *n*]) [Function File]
rsys = repsys (*sys*, *m*) [Function File]

Form a block transfer matrix of *sys* with *m* copies vertically and *n* copies horizontally. If *n* is not specified, it is set to *m*. `repsys (sys, 2, 3)` is equivalent to `[sys, sys, sys; sys, sys, sys]`.

2.21.7 strseq

strvec = strseq (*str*, *idx*) [Function File]

Return a cell vector of indexed strings by appending the indices *idx* to the string *str*.

```
strseq ("x", 1:3) = {"x1"; "x2"; "x3"}
strseq ("u", [1, 2, 5]) = {"u1"; "u2"; "u5"}
```

2.21.8 test_control

test_control [Script File]

Execute all available tests at once.

The Octave control package uses the Uses SLICOT library, Copyright (c) 1996-2025, SLICOT, available under the BSD 3-Clause (License and Disclaimer). SLICOT needs BLAS and LAPACK libraries which are also prerequisites for Octave itself. In case of failing tests, it is highly recommended to use Netlib's reference BLAS and LAPACK for building Octave. Using ATLAS may lead to sign changes in some entries of the state-space matrices. In general, these sign changes are not 'wrong' and can be regarded as the result of state transformations. Such state transformations (but not input/output transformations) have no influence on the input-output behaviour of the system. For better numerics, the control package uses such transformations by default when calculating the frequency responses and a few other things. However, arguments like the Hankel singular Values (HSV) must not change.

Appendix A License Information

A.1 License and Copyright for parts fo the Control Package

The following list provides license and copyright information for file locations in the package's Git repository (*Git*) and in the distributed package archive `control-x.y.z.tar.gz` (*Dist*).

- All files except those specified below:
  Copyright see individual files
  License GPLv3+ (see Sec. 2)
- `slicot-reference` (*Git*, git submodule), `src/slicot` (*Dist*):
  Copyright (c) 1996-2025, The SLICOT Team
  License BSD 3-Clause (see Sec. 3)
- `slicot-reference/src/lapack_aux` (*Git*, git submodule), src/slicot/lapack_aux (*Dist*):
  Copyright (c) 1992-2025 The University of Tennessee, The University of Tennessee Research Foundation
  Copyright (c) 2000-2025 The University of California Berkeley
  Copyright (c) 2006-2025 The University of Colorado Denver
  License BSD 3-Clause (see Sec. 3)

A.2 GPLv3+ License

Version 3, 29 June 2007

A.2.1 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

A.2.2 Terms and Condition

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
   b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
   d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
   e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
   f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

   All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

   If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

   Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

A.2.3 End of Terms and Conditions

A.2.3.1 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

A.3 BSD 3-Clause License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE

Function Index